The Generalized Chord Diagram Expansion

Dissertation
zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)

im Fach Mathematik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von

Dipl.-Math. Markus Hihn

Präsident der Humboldt-Universität zu Berlin:

Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät:

Prof. Dr. Elmar Kulke

Gutachter:

1. Prof. Dr. Dirk Kreimer
2. Prof. Dr. Matthew DeVos
3. Prof. Dr. Loïc Foissy

eingereicht am: 3.12.2015
Tag der Verteidigung: 18.4.2016

To the people who believed and still believe in me, especially to my grandfather

Michael Hihn sen. (27.3.1921-14.5.2015).

Contents

1 Introduction and outline

1.1 Overview

> "If that's what's written, then that's what's written"
> - Ferengi Rule of Acquisition

In Quantum field theory Dyson-Schwinger equations are integral (fix point) equations that come from self insertion properties of Feynman graphs: The crucial property that is needed is that the Feynman amplitudes X^r of a given residue (a residue means a given external leg structure here) can be obtained by inserting a combinations of other amplitudes into itself. This leads to a recursive term by term expansion as done in perturbation theory. Dyson-Schwinger equations are in general hard to control. There are different reasons for this:

1. In general, for every coefficient of a Dyson-Schwinger equation there are a finite number of Feynman-Integrals to calculate, which need renormalization, regularization and a good knowledge of special functions that seems to be never enough. For example in ϕ^4 lower coefficients need a very good knowledge of polylogarithms, however, for higher coefficients it is still not known what sort of special functions are involved.

2. Controlling the combinatorics of appearing Feynman graphs: The combinatorics even in restricted or toy theories is often highly non trivial, as we will shall see later.

The following Dyson-Schwinger equation is a start, because it has both points better under control:

$$G(x,L) = 1 - xG(x,\partial_{-\rho=0})^{-1}\tilde{F}(\rho) \text{ where } \tilde{F}(\rho) = (e^{-L\rho}-1)\sum_{k\geq 0} a_k \rho^{k-1}$$

It is a generalization of a Dyson-Schwinger equation that was studied by Kreimer and Broadhurst, where they studied the anomalous dimension of a fermion field with a Yukawa interaction $g\bar{\psi}\sigma\psi$ whose rainbow approximation at $d_c = 4$ (see [2]).

Doing the Ansatz

$$G(x,L) = 1 - \sum \gamma_k(x) L^k \text{ with } \gamma_k(x) = \sum_{l\geq} \gamma_{kl} x^l$$

we can calculate (in the world of formal series!) the first $\gamma_{kl}(-1)^k/k!$ by doing the derivatives and comparing both sides:

$$\begin{aligned}
\gamma_{11} &= -a_0 \\
\gamma_{12} &= -a_1 a_0 \\
\gamma_{22} &= \frac{1}{2} a_0^2 \\
\gamma_{13} &= -3a_0^2 a_2 - a_1^2 a_0 \\
\gamma_{23} &= 2a_1 a_0^2 \\
\gamma_{33} &= -\frac{1}{2} a_0^3 \\
\gamma_{14} &= -a_1^3 a_0 - 15 a_0^3 a_3 - 11 a_1 a_0^2 a_2 \\
&\dots
\end{aligned}$$

Setting all $a_i = 1$ and considering all terms up to $\gamma_{1,6}$ which we do not list (but which can be calculate easily), we obtain for $\gamma_1(x) = -x - x^2 - 4x^3 - 27x^4 - 248x^5 - 2830x^6 \dots$ which shows evidence that $-\gamma_1$ might be the generating function for rooted connected chord diagrams (see OEIS A000699 for more coefficients)! You might ask yourself, where the Feynman graphs and integrals are. The answer is that they are already done. The parameter ρ comes from the analytic regularization (see [16] p. 15pp.). x is the coupling constant and L is a logarithm of the the momentum scale: $L := \log(\frac{q^2}{\mu^2})$. The recursive form comes from a Mellin transform trick which can be applied on special Greens functions. The a_k are a generalization of $\frac{1}{\rho(\rho-1)}$ which is all $a_k = 1$. The $a_k = 1$ case was studied in [2] and generalized by Karen Yeats for arbitrary a_k when $s = 2, N = 1$ in [7] . The answer is described in terms of rooted connected chord diagrams, denoted by $\mathcal{R}$ (rooted connected chord diagrams are introduced in Chapter 2) by

$$G(x,L) = 1 - \sum_{i\geq 1} \frac{(-L)^i}{i!} \sum_{C\in\mathcal{R}, b(C)\geq i} x^{|C|} \underline{\hat{a}}_C a_{b(C)-i} \tag{1.1.1}$$

$\underline{\hat{a}}_C$ is a monomial build from a chord diagram C. $b(C)$ is a positive integer and it is given by the so called base chord. $i = 1$ gives the anomalous dimension and is, due the renormalization group equation, all we need for the whole Greens function. This reduces a quite general Dyson-Schwinger equation to the combinatorial study of rooted connected chord diagrams (which is not as simple as one would guess). If we set all $a_k = 1$, then it

also makes clear what numbers Dirk Kreimer and David Broadhurst in [2] observed. In the following we want to "solve" a bigger class of Dyson-Schwinger equations, namely for a given $s \in \mathbb{N}_{\geq 2}$ the more general case:

$$G(x,L) = 1 - \sum_{k \geq 1} x^k G(x, \partial_{-\rho=0})^{1-sk} \tilde{F}_k(\rho) \tag{1.1.2}$$

where

$$\tilde{F}_k(\rho) = (e^{-L\rho} - 1) \sum_{l \geq 0} a_{k,l} \rho^{l-1},\ k \in \mathbb{N}$$

Note that we do not require $a_{k,l} \neq 0$. We define

$$N := \max\{k \in \mathbb{N} : F_k(\rho) \neq 0\} \text{ and } N := \infty \text{ if all } F_k \neq 0$$

Thus the case we summarized shortly before was $s = 2, N = 1$. It turns out that the more general case can be also described in terms of rooted chord diagrams but this time decorated ones with multiplicities that can be described by a non-trivial weight construction:

$$G(x,L) = 1 - \sum_{i \geq 1} \frac{(-L)^i}{i!} \sum_{C \in \mathcal{R}^{dec}, b(C) \geq i} \omega_C a_{\hat{C}} a_{b(C)-i} x^{||C||} \tag{1.1.3}$$

This looks similiar to 1.1.1, we explain shortly the difference: Let $d_1, \ldots, d_m$ be powers of x of the initial Dyson-Schwinger equation which have non zero $\tilde{F}_.$, we choose the decorations $\mathcal{D} = \{d_1 \ldots d_m\} \subset \mathbb{N}$, each decoration may be placed at any chord on a $C \in \mathcal{R}$ yielding an decorated chord diagram $C^{dec} \in \mathcal{R}^{dec}$. Where $|C|$ was the numbers of chords before, it is now the sum of the decorations: $||C|| := \sum d_k$. To construct a monomial we first need to look at the terminal chords (terminal chords are explained in section 2.1) of C, it is a finite non-empty set of size $M+1$ (which depends on the chord diagram, of course)

$$\text{ter}(C) = \{b(C), t_1, \ldots, t_M\}\ ;\ b(C) < t_1 < \ldots < t_M$$

The monomial $\underline{\hat{a}}_C$ is depending only on $t_1, \ldots, t_M$ In contrary, the weight ω_C depends only on the tree structure of a rooted chord diagram (how to associate an unique tree to a rooted chord diagram will be explained in sections 2.3).

That this construction works indeed for all our Dyson-Schwinger equation 1.1.3 is the main result of this thesis.

1.1.1 A short introduction to Quantum Field Theory

> "Hear all, trust nothing"
> - Ferengi Rule of Acquisition no. 190

> "But if we dont't have a clue to find them, we are lost. That's it: We have to get lost ourselves. We are going to build a get lost machine. A get lost machine was deliberatily designed so that you never knew where you go." Dr. Snuggles, "The Great Disappearing Mystery"

Quantum field theory is the unification of special relativity with quantum mechanics to understand the behaviour of subatomatic particles like photons, electrons, etc. While physicists apply quantum field theory with enormous success to experiments and observations in the particle colliders (like the discovery of the Higgs boson recently), there are a lot of open questions. For example from a mathematical viewpoint: How to make this theory well defined (Haags theorem, products of distributions, path integral, etc.)?
A lot of mathematical techniques were developed to attack these problems (the theory of algebraic quantum field theory, non-commutative geometry, etc.).
From a physicist viewpoint: How to get better accuracy in the comparison with experiments? While the use of newer mathematical QFT models for physicists calculations is still limited, a pragmatic approach that physicists regularly use is the the scattering theory, which calculate the first coefficients of the so called S-Matrix. The scattering theory is only an approximation to the real world, it corresponds to the real world experiments, with very good accuracy, though. However, this is still very complicated, the calculations need the help of Feynman diagrams to keep track of very complicated integrals and a lot of intrinsic calculations were used. This made it hard for mathematicians to understand what was going on in the scattering theory of Quantum Field Theory. Kreimer observed that some techniques which were already known in physics can be interpreted through algebraic structures: The formulation of the BPHZ mechanism by a Hopf algebra, led to enormous interest in number theory and algebra for Quantum Field Theory (the recent success of the parametric representation of Feynman integrals, the theory of combinatorial Dyson-Schwinger equation and mathematically defined Feynman rules is another story that is just in its beginnings). There are also applications for physics calculation see for example (see [2]).
There are a lot of introductory texts on Quantum Field Theory.[1] While it is not a problem to define non-interacting particles (Wightman's Axioms), it is hard to define what particle interaction should be. Sadly, as mentioned before this exactly what physicists want to have! Roughly speaking, the S-Matrix is a formal series, where the n-th summand

[1] for physicists one of standard introductory text books to mention are for example [8] or [11]), for mathematicians the book of Folland([4]) is a very good start

is given by an integration over n closed loops given by quantum fields.[2] A quantum field looks like:

$$\phi(x) = \int \frac{d^3\mathbf{p}}{(2\pi)^3} \frac{1}{\sqrt{2\omega_{\mathbf{p}}}} \left(e^{-i\omega_{\mathbf{p}}t + i\mathbf{p}\cdot\vec{x}} a(\mathbf{p}) + e^{i\omega_{\mathbf{p}}t - i\mathbf{p}\cdot\vec{x}} a^\dagger(\mathbf{p})\right) \text{ where } x = (\vec{x}, t)$$

where $a^\dagger$ is called creation operator and a annihilation operator, which both are distribution valued. In a real-world Quantum field theory these operators depend on a momentum (which is a real vector), a spin state and a particle species (bosonic/fermionic). Luckily, products of creation, annihilation operator can be evaluated, depending on their momentum, spin state and particle species. This leads to the application of Wick contractions and in the end to the development of Feynman graphs. This being said, Feynman graphs encode a complicated nested integral of quantum fields, that can be ill-defined. To cure singularities of those integrals, renormalization techniques are used and the values obtained after this process are the values of renormalized Feynman rules.

The before mentioned Hopf algebra structures allow to study fixpoint equations in the Hochschild cohomology of the Hopf algebra of Feynman graphs. By applying Feynman rules, they yield Dyson-Schwinger equations which we study here.

Classically, Dyson Schwinger Equations (in short DSE) are derived from the path integral formalism. Since this formalism is a mathematical problem itself, we can just leave it as inspiration for defining mathematical well defined objects. This is were the Hopf algebra of Feynman graphs come into play. We distinct between combinatorial Dyson Schwinger equations which are fix point equations in the Hochschild cohomology of H_{FG}, H_{RT}[3] and analytic Dyson Schwinger equation which are fix point equations over the ring formal series $\mathbb{C}[[x, L]]$. The Hopf algebra of decorated trees is a good model for true Quantum field theories, because the number of primitive graphs is countable. They are related by renormalized Feynman rules. A class of combinatorial Dyson-Schwinger equations in H_{RT} is given by $s \in \mathbb{Z}_{\geq 0}$:

$$X = 1 - \sum_{k \geq 1} x^k B_+^k(X^{1-sk}(x))$$

where B_+^k is a Hochschild-1-Cocycle in the Hopf algebra of rooted trees: It grafts a forest (that is a product of trees) under a root that is decorated by the integer k. Under certain renormalized Feynman rules (see [16] it leads to the analytic Dyson-Schwinger equations 1.1.2, that we will study here. This class of combinatorial Dyson-Schwinger equations

[2] To develop it accurate a lot more techniques are needed like the time ordering operator, the distinction of non-interacting and interaction Hamiltonian, etc.

[3] Here H_{FG} denotes the Hopf algebra of Feynman graphs of some Quantum Field Theory and H_{RT} the Hopf algebra of decorated rooted trees. For an introduction into both Hopf algebras, see [6]

is a simplification of systems that are relevant in quantum field theory (see Example 3.4 in [16]). Some solutions of combinatorial Dyson-Schwinger equations form a sub Hopf algebra. In the Hopf algebra of rooted trees the combinatorial Dyson-Schwinger equations that form sub Hopf algebras are described by the following theorem (for more details see Theorem 4 of [3]).

Theorem 1.1.1. *Let f be a formal power series over $\mathbb{C}$ with $f(0) = 1$, then the following assertions are equivalent:*

1. *The coefficients of the solution of $X = B_+(f(X))$ span a sub Hopf algebra of H_{RT}*

2. *There exists $(\alpha, \beta) \in \mathbb{C}^2$ such that*

$$(1 - \alpha\beta h)f'(h) = \alpha f(h)$$

3. *There exists $(\alpha, \beta) \in \mathbb{C}^2$ such that*

$$f(h) = \begin{cases} 1 & \alpha = 0 \\ e^{\alpha h} & \beta = 0 \\ (1 - \alpha\beta h)^{-1/\beta} & \alpha\beta \neq 0 \end{cases}$$

While most of the situations that happen in physics can be encoded in the Hopf algebra of decorated trees, the Hopf algebras of Feynman graphs are more complicated. For example, the Hochschild 1-Cocycles in the Hopf algebra of decorated trees are the grafting operators B_+^k, that we mentioned before. In the Hopf algebra of Feynman graphs there are similar grafting operators $B_+^{k,r}$ to primitive graphs with loop number k and residue r (see [5], [16]). However, to ensure that they fulfill the Hochschild 1-Cocycle property (which is crucial for the formulation of Feynman rules on these Hopf algebras)

$$\Delta B_+ = (1 \otimes B_+)\Delta + B_+ \otimes 1$$

one has to consider quotients over some Hopf ideals described for example by the Slavnov-Taylor identites in the Hopf algebra of Feynman graphs of QCD (see [13]).

An analytic DSE is an equation that comes from a combinatorial DSE after applying some renormalized Feynman rules. When using analytic regularization it can sometimes be transformed into the following form:

$$G(x, L) = 1 + \text{sign}(s) \sum_k G(x, \partial_{-\rho=0})^{sk} F_k(\rho)$$

where $G(x, L)$ is a formal power series

$$G(x, L) = 1 + \text{sign}(s) \sum_{k,l \geq 1} \gamma_{k,l} x^k L^l$$

the parameter x is called the coupling constant and $L = \log(q^2/\mu^2)$ is a logarithm of mass scales. The formal power series $\gamma_l(x) = [L^l]G(x, L)$ can be constructed recursively from $\gamma_1(x)$ by the renormalization group equation

$$\gamma_k = \gamma_1(1 + \text{sign}(s)x\partial_x)\gamma_{k-1}$$

. For its outstanding importance $\gamma_1(x)$ has an own name and is called the anormalous dimension[4]. It describes the β-function (see [12] and is important in physics This renormalization group equation was proved in Theorem 4.10 of [16].

Analytic Dyson Schwinger equations give a possibly infinite system of differential equation of the self energy in quantum field theories and therefore are a good way to approximate the Greens function.

Starting on the combinatorial side, the problem that arises first is a combinatorial explosion, even on some Dyson-Schwinger equations over the Hopf algebra of rooted trees (which is a good approximation to the Hopf algebra of Feynman graphs). However, some solutions (to arbitrary order) are well known for a certain class of cDSE. A messy mathematical problem is the transformation from combinatorial to analytic Dyson-Schwinger equation, since an unsatisfied interchange of summation and integration comes into play. However, besides that the analytic side has a nice combinatorial flavour of its own.

The perturbative Greens function of a Dyson-Schwinger equation can be solved by comparing coefficients and resolving it term by term. For example, let us try to get the first coefficients from

$$G(x, L) = 1 - xG(x, \partial_{-\rho=0})^{-1} F(\rho, L)$$

where $F(\rho, L) = (e^{-L\rho} - 1)\frac{1}{\rho} \sum_{k \geq 0} a_k \rho^k$ for some real numbers a_k. First, we do the Ansatz:

$$G(x, L) = 1 - \sum_{k \geq 1} \gamma_k(x) L^k \text{ where } \gamma_k(x) := \sum_{l \geq k} \gamma_{k,l} x^l$$

Thus, we can use the geometric series (this makes sense in the topology of formal series) to get:

$$G(x, \partial_{-\rho=0})^{-1} = \sum_{n \geq 0} \left(\sum_{k \geq 1} \gamma_k(x) \partial^k_{-\rho=0} \right)^n$$

[4]here the name dimension comes from the slang of physicists calling different physical units dimension

So we need to calculate:

$$\partial^n_{-\rho=0}\left[F(\rho,L)\right] = \partial^n_{-\rho=0}\left[\sum_{l\geq 1,k\geq 0}\frac{(-L)^l}{l!}a_k(-\rho)^{l+k-1}\right] =$$
$$= \sum_{l\geq 1,k\geq 0,l+k-1=n}\frac{(-L)^l}{l!}a_k n! = \sum_{0\leq k\leq n} a_k\frac{n!}{(n-k+1)!}(-L)^{n-k+1}$$

Note that this can be rewritten nicely:

$$\partial^n_{-\rho=0}(e^{-L\rho}-1)F(\rho) = F(-\partial_L)(L^n) \text{ where } F(\rho) = \frac{1}{\rho}\sum_{k\geq 0}a_k\rho^k$$

Back to the equation we can calculate the first coefficients of γ_1 immediately:

$$G(x,L) = 1 - \left(\sum_{n\geq 0}\left(\sum_{k\geq 1}\gamma_k(x)\partial^k_{-\rho=0}\right)^n\right)F(\rho,L)$$
$$= 1 - x\left(\mathrm{id}_{-\rho=0} + \sum_{k\geq 1}\gamma_k(x)\partial^k_{-\rho=0} + \left(\sum_{k\geq 1}\gamma_k(x)\partial^k_{-\rho=0}\right)^2 + \ldots\right)F(\rho,L)$$

And since

$$x\left(\sum_{k\geq 1}\gamma_k(x)\partial^k_{-\rho=0}\right)F(\rho,L) = x\sum_{n\geq 1}\gamma_n(x)\sum_{0\leq k\leq n}a_k\frac{n!}{(n-k+1)!}(-L)^{n-k+1}$$
$$= x^2\gamma_{1,1}\left(\frac{1}{2}a_0L^2 - a_1L\right) + O(x^3)$$

we get

$$G(x,L) = 1 + L(xa_0 + x^2a_0a_1) + O(xL^2) + O(x^3L)$$

2 Chord diagrams

2.1 Rooted connected chord diagrams and Insertion Trees

> "Alles mit Bildern. Zeig doch mal dem Herrn die Bilder!"
> aus "Du und dein Koerper" von Loriot

Sometimes, a chord diagram is defined to be a 3- regular graph where each vertex lies on a circle. For this we would need to fix a plane embedding such that all edges that are not in the circle are inside the circle. Also, we would need to identify homeomorphic transformations. While this is the geometric picture we have in mind, we first use an algebraic definition. This will make it easier for us, to talk about crossings and the root share decomposition without having any topological concerns. However, later on we will switch to the geometric perspective whenever it is more helpful.

Definition 2.1.1 (Rooted chord diagram)**.** A rooted chord diagram D of size n is a fix point free involution $D \in S_{2n}$. That is a permutation such that $D^2 = \mathrm{id}_{S_{2n}}$ with $D(i) \neq i$ for all $i = 1 \dots 2n$. Equivalently it is a permutation that can be written as disjoint transpositions without fixed points:

$$D = (x_1 y_1)(x_2 y_2) \cdots (x_n y_n)$$

where $x_1 < \ldots < x_n$ and $x_i < y_i$ for all $i = 1 \dots 2n$. Each transposition is called a chord and $(x_1 y_1)$ is called the root chord.

To justify the name rooted chord diagram, draw a circle and mark $2n$ vertices on it. Choose a distinct vertex as the root and label the vertex 1, enumerate the vertices counter-clockwise and draw a chord between vertex x_i and vertex y_i for each transposition.

We say that a chord $(x_i y_i)$ crosses a chord $(x_j y_j)$ if

$$x_i < x_j < y_i < y_j \text{ or } x_j < x_i < y_j < y_i$$

.

To be able to define the property of being connected on a chord diagram, we introduce intersection graphs now:

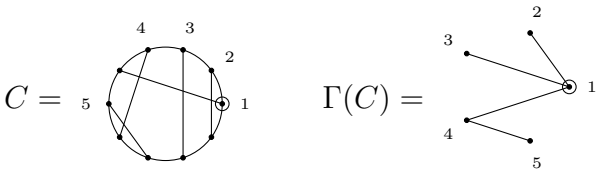
Definition 2.1.2 (Labeled intersection graph)**.** Let $c_1, \ldots, c_n$ be a fixed labeling of the chords. The labeled intersection graph of a chord diagram C with respect to that labeling is a vertex-labeled simple graph $\Gamma(C) = (V, E)$ given by

$$V = \{1, \ldots, |C|\} \qquad E = \{(i, j) : c_i \text{ crosses } c_j\}$$

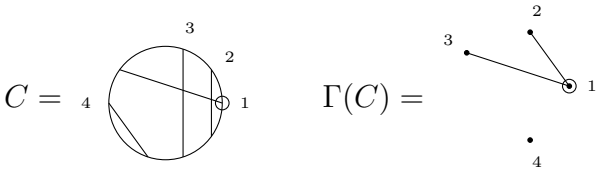
If there is no risk of confusion, we always identify the labelings with the underlying objects (i.e. vertices, edges, chords, etc.).

If the labeled intersection graph of a rooted chord diagram is connected, we call the chord diagram connected otherwise we call it disconnected. The chord diagrams of a disconnected chord diagram D induced by the connected components of the intersection graph are called the connected components of D. We denote the set of all rooted connected chord diagrams $\mathcal{R}$.

Example 2.1.3. A rooted connected chord diagram C with its intersection graph $\Gamma(C)$:



1 crosses $2, 3, 4$ (also vice versa: 2 crosses 1, etc.) and 4 crosses 5. Removing for example the chord 4 results in a disconnected intersection graph, so this chord diagram is called disconnected:



Definition 2.1.4 (Intersection order)**.** The intersection order of a rooted chord diagram C is defined recursively by the following pseudo code:

```
intersection_order(k, C) {
  m := root(C)
  label(m) := k
  k := k + 1
  if |C| != 1 then
  foreach D := connected_components(C \ m) traversed counter clockwise
  {
    intersection_order(k,D)
```

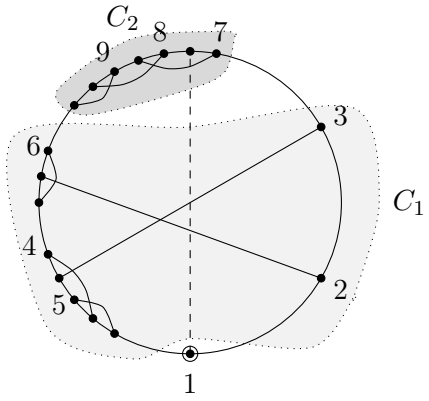
```
      k := k + |D|
    }
  }
```

and start the procedure with

```
intersection_order(1,C).
```

From now on we will stick to the intersection order, because it is crucial for later constructions.

Example 2.1.5. The picture below shows an example of connected components and the intersection order. The chord diagram will be a specific example of a canonical class which we will define later (see Definition 4.2.1). This chord diagram is called $CW_3(3,1,2)$ and its two connected components after removing the root chord are indicated by gray color, the lighter one is the first connected component, the darker one the second connected component. The numbers near the circle indicate the chord's position in the intersection order.



If we want to restrict $\mathcal{R}$ to only connected chord diagrams with n chords, we will write $\mathcal{R}(n)$. To a rooted connected chord diagram the terminal set $\mathrm{Ter}(C)$ is defined as in [7]:

$$\mathrm{Ter}(C) = \{c \in C : \text{ there is no higher labeled chord } d \text{ } s.t. \text{ } c \text{ crosses } d\}$$

Furthermore, the subset of rooted connected chord diagrams restricted to a terminal S. We notice that every chord diagram has exactly max S chords.

Proposition 2.1.6. *The rooted chord diagram is uniquely determined by its labeled intersection graph.*

Proof. Obviously, to every chord diagram there is a labeled intersection graph. For uniqueness, suppose there are two different chord diagrams C_1, C_2 with intersection graph $\Gamma(C_1) = \Gamma(C_2)$. W.l.o.g. there is a crossing in C_1 which is not in C_2, however, this means that there is an edge in $\Gamma(C_1)$ that is not in $\Gamma(C_2)$, i.e. $\Gamma(C_1) \neq \Gamma(C_2)$. □

Of course, the latter result does not hold if we drop the label. For a labeled intersection graph Γ the terminal vertices

$$\mathrm{Ter}(\Gamma) = \{v \in V(\Gamma) : v \text{ has only smaller neighbours in } \Gamma\}$$

clearly correspond to the terminal chords.

Definition 2.1.7. For a natural number n we define $\mathrm{Ter}(n)$ to be the family of all terminal sets possible for $C \in \mathcal{R}(n)$

$$\mathrm{Ter}(n) := \{\mathrm{Ter}(C) : C \in \mathcal{R}(n)\}$$

It is actually not hard to see what this family looks like.

Proposition 2.1.8. *Let $n \neq 1$ be a positive integer, then*

$$\mathit{Ter}(n) = \Big\{\{k_1, \ldots, k_l\} \in \mathbb{N}^l : \ 2 \leq k_1 < \ldots < k_l = n, \ l = 1, \ldots, n-1\Big\}.$$

If $n = 1$, then $\mathit{Ter}(1) = \Big\{\{1\}\Big\}$

Proof. The case $n = 1$ is trivial, because there is only one chord which is terminal. Note that if $n \neq 1$, the root is never terminal, so the least terminal chord is 2 for any chord diagram $|C| > 2$. The case $n = 2$ is trivial also, because there is only one rooted connected chord diagram, which has clearly only 2 as terminal chord. Let $n \neq 2$, the statements holds true for $\mathrm{Ter}(n-1)$ by induction. Let $C \in \mathrm{Ter}(n-1)$, wedging a new chord around the root (i.e. adding a chord $(1\ 2n)$) or wedges a new chord on a chord $(x\ y)$ (i.e. adding a $(y-1)(y+1)$) of C results in any possible set of $\mathrm{Ter}(n)$. Adding a chord in any other way does not change the possible sets, because either we cross terminal chords that would create a new terminal, which would not yield any new terminal chord, that we could not construct with wedging, or we didn't construct a new terminal which will shift the labels of the old terminals by 1 which is still in any possible set of $\mathrm{Ter}(n)$. □

An alternate proof can be done via induction and the help of 2.2.9.

2.2 Root share decomposition and Insertion trees

Beginning at the root we order the vertices of the chord diagram counterclockwise. Let's call the edge on the circle between the vertex k and the vertex $k+1$ the kth interval of the chord diagram. The interval between vertex $2|C|$ and the root is called the 0th interval. We can insert a rooted connected chord diagram $C \in \mathcal{R}(n)$ into another chord diagram $D \in \mathcal{R}(m)$, by placing the root of C into the 0-th interval of D and the other end of

the root chord and the rest of the chord diagram in the k-th interval of D. This gives us an insertion operation for every $k = 1, \ldots, 2|D| - 1$. To be more specific all vertices of C except the root will be shifted in the new chord diagram by k, the root of D will be shifted by one because it is the neighbour vertex of the root of C and all other vertices of D (except the vertices with vertex labeling smaller or equal k) will be placed after the last vertex of C, so their vertex labels are shifted by $2n$. The adjacency of the chords will be derived by C and D. To make this definition precise, we define:

Definition 2.2.1 (Insertion operation of rooted chord diagrams)**.** Let $C \in \mathcal{R}(n), D \in \mathcal{R}(m)$ and

$$C = (x_1 y_1) \ldots (x_n y_n)$$
$$D = (x'_1 y'_1) \ldots (x'_m y'_m)$$

there underlying permutation then for each $k = 1, \ldots, 2m - 1$ define $C \circ_k D \in \mathcal{R}(n + m)$ by the following permutation:

$$(x_1, y_1 + k)..(x_n + k, y_n + k)(H_{n,k}(x'_1), H_{n,k}(y'_1))..(H_{n,k}(x'_m), H_{n,k}(y'_m))$$

where $H_{n,k}$ is defined to be:

$$H_{n,k}(x) = \begin{cases} x + 1 & \text{if } x \leq k \\ x + 2n & \text{otherwise} \end{cases}$$

$C \circ_k D$ is indeed a rooted connected chord diagram of size $n+m$, because every integer from one to $2(n + m)$ is appearing exactly once in the transpositions, it is fix point free and the labeling shift does not destroy any crossings but the root chord of C creates new crossings with at least one chord of D.

Example 2.2.2.

$$(1, 4)(2, 6)(3, 5) \circ_2 (1, 4)(2, 5)(3, 6) = (1, 6)(2, 10)(3, 11)(4, 8)(5, 7)(9, 12)$$

Remark 2.2.3. Note that the insertion operation is highly non associative and non commutative. If $C \circ_k D$ is defined, $D \circ_k C$ may not be defined. For example, if C is a chord diagram with only one chord, then $D \circ_k C$ is not defined for $k \geq 2$ because there is only one insertion interval in C by definition.

In the following definition of the root share decomposition, we will need to decompose a chord diagram. Since by our definition chord diagrams are certain permutations, we need to define for an expression

$$A = (a_1 a_2) \ldots (a_{2n-1} a_{2n})$$

where a_k ($k = 1..2n$) are arbitrary distinct natural numbers, the associated normalized involution norm(A) by

$$\mathrm{norm}(A) = \big(\sigma(1)\sigma(2)\big)\ldots\big(\sigma(2n-1)\sigma(2n)\big)$$

where σ sorts the integers, i.e. $\sigma \in S_{2n} : a_{\sigma^{-1}(1)} < a_{\sigma^{-1}(2)} \ldots < a_{\sigma^{-1}(2n)}$

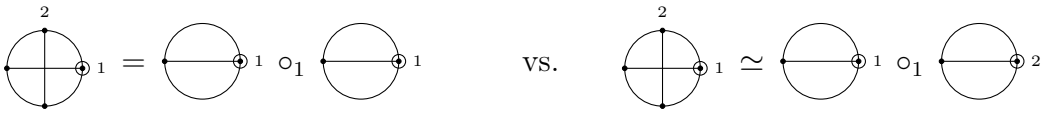
Example 2.2.4. Let $A = (13)(28)(57)$, then $\mathrm{norm}(A) = (13)(26)(45)$

Definition 2.2.5 (Root share decomposition)**.** Let $C \in \mathcal{R}$ with $|C| > 1$, there exists a unique i such that

$$C = C' \circ_i C'' \text{ where } C' = \mathrm{norm}(C \setminus C_1), C'' = \mathrm{norm}(C_1)$$

and C_1 is the first connected component after removing the root temporarily. Note that $C \setminus C_1$ is always connected. This unique decomposition is called the root share decomposition of C.

That this decomposition is well defined, i.e. there is such an unique i, will be shown in Proposition 2.2.6. Suppose we have a root share decomposition $C = C' \circ_i C''$ then we write $C \simeq C' \circ_i C''$ if C', C'' are considedered with the labeling induced by C, which is not an appopriate labeling for C', C'' but a shift of it. This is an abuse of notation but it helps us keep our formulas compact. So for example:



Proposition 2.2.6. *Let $D = D' \circ_k D''$ its corresponding root share decomposition and*

$$D' \simeq (x'_1 y'_1)\ldots(x'_m y'_m)$$

the corresponding permutation with vertex labels induced from D, then the k is given by

$$k = \begin{cases} y'_1 - 2 & \text{if } |D'| = 1 \\ x'_2 - 2 & \text{otherwise} \end{cases}$$

Proof. There are two cases two consider:

1. D' consists only of one chord, then that is root chord chord and its other end indicates the insertion place. The normalization to D'' gives the shift of -2.

2. D' consists of more than one chord. The chords after the root chord are coming from the second connected component of D after removing the root. However, D'_1

is inserted into the interval before the vertex x_2' and the count of the interval starts with 0, therefore we have $k = x_2' - 2$.

□

The last proposition justifies that such a unique k exists.

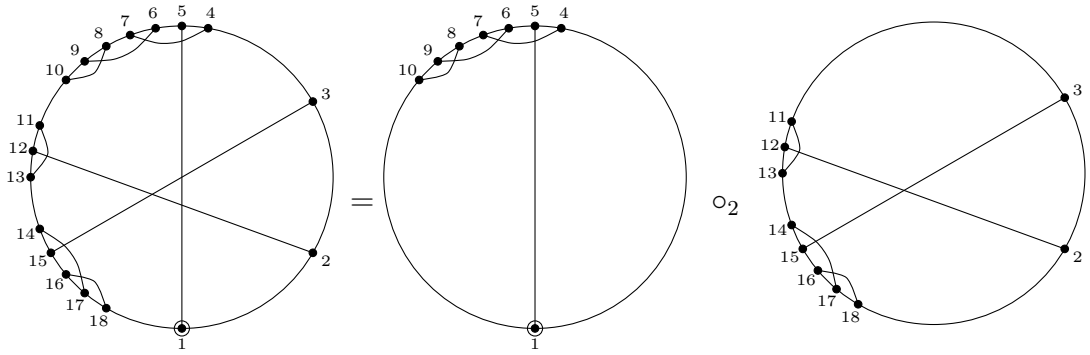
Example 2.2.7. Consider the chord diagram $CW_3(3, 1, 2)$ as a permutation. With the notation of the last proposition we have

$$D = (1, 5)(2, 12)(3, 15)(4, 7)(6, 9)(8, 10)(11, 13)(14, 17)(16, 18)$$

and on the right hand side

$$D' = (1, 5)(4, 7)(6, 9)(8, 10),\ D'' = (2, 12)(3, 15)(11, 13)(14, 17)(16, 18)$$

thus $k = 4 - 2 = 2$. The picture below shows the chord diagrams:



Remark 2.2.8. Every $C \in \mathcal{R}(n)$ can be decomposed into $C^1, \ldots, C^n \in \mathcal{R}(1)$ by applying recursively $n - 1$ insertion operations. This decomposition is unique.

For the next proposition we introduce the following notation: Let $A \subseteq \mathbb{Z}$ and $n \in \mathbb{Z}$, then $A + n$ is defined to be the set $\{a + n \ : a \in A\}$.

Proposition 2.2.9. *Let $C = C_1 \circ_k C_2$ for some $k = 1, \ldots, 2|C_2| - 1$ then*

$$\mathit{Ter}(C) = (\mathit{Ter}(C_1) + |C_2|) \cup (\mathit{Ter}(C_2) + 1)$$

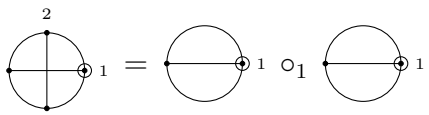
Proof. Let $C = C_1 \circ_k C_2$. If $|C_1| = 1$, then $\mathrm{Ter}(C_2)$ gets shifted by 1 but also

$$\mathrm{Ter}(C_1) + |C_2| = |C_2| + 1 \subseteq \mathrm{Ter}(C_2) + 1$$

since the last chord is always terminal. Now consider $|C_1| \neq 1$, then by definition of the insertion operation the terminals stay the same only modified by a labeling shift (the

chord structure of C_1 and C_2 stays the same for the terminals and the only new crossings that can appear are crossings with the root which is not terminal). So it is left to show that $\mathrm{Ter}(C_1)$ is shifted by $|C_2|$ and $\mathrm{Ter}(C_2)$ is shifted by 1. The first holds true, because after removing the root of C the remaining chords of C_1 form connected components that lie counterclockwise after C_2. The second holds true, because the newly inserted root shifts all labels of C_2 by one. □

Example 2.2.10. Let C_1, C_2 be the two trivial chord diagrams and consider $C_1 \circ_1 C_2$:



hence $\mathrm{Ter}(C_1) = \mathrm{Ter}(C_2) = 1$ and so the rooted connected chord diagram with two chords $C_1 \circ_1 C_2$ has indeed $\mathrm{Ter}(C_1 \circ_1 C_2) = (\{1\} + 1) \cup (\{1\} + 1) = \{2\}$.

The previous proposition makes it possible for us to reconstruct the number of rooted chord diagrams with a fixed terminal set. However, the shifts in the labelings causes some trouble to reconstruct a general recursive formula. We will explain in a short example how the reconstruction is possible in a specific case.

Example 2.2.11. Let's calculate the number $\mathcal{R}_{\{3,5\}}$ of rooted connected chord diagrams with terminal set $\{3, 5\}$. If we insert C_1 with $\mathrm{ter}(C_1) = \{3\}$ at any place into a diagram C_2 with $\mathrm{ter}(C_2) = \{2\}$, we get a chord diagram with the wanted terminal sets. The same holds true for C_1 with $\mathrm{ter}(C_1) = \{1\}$ and C_2 with $\mathrm{ter}(C_2) = \{2, 4\}$. We notice that these two cases are the only ones for $\{3, 5\}$. So by looking up $\mathcal{R}_{\{3\}}$ and $\mathcal{R}_{\{2,4\}}$ we obtain

$$|\mathcal{R}_{\{3,5\}}| = (2 \cdot 3 - 1)|\mathcal{R}_{\{3\}}| + (2 \cdot 4 - 1)|\mathcal{R}_{\{2,4\}}| = 30$$

To every rooted connected chord diagram C, there is a unique tree $T(C)$ that is constructed recursively by the root share decomposition. To transport the root share decomposition to trees, we introduce the

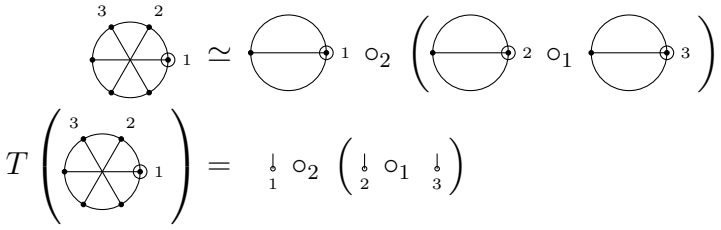
Definition 2.2.12 (Insertion operation on rooted plane trees)**.** Let T, T' be rooted plane trees where the edges are labeled in pre-order (in our definition the root has a virtual hook edge which we label always by 1). The rooted plane tree is $T \circ_k T'$ is defined by putting the hook edge of T into the edge k of T' and placing T as left sub tree and the subtree of T' rooted at the bottom end of k as right subtree.

Example 2.2.13. Let $S =$  and $T =$  then $S \circ_2 T =$ 

2.3 Branch-left vectors

If we consider the chord diagram expansion for the analytic Dyson-Schwinger equation (1.1.2) for $s \neq 2$ and $N \geq 2$, each chord diagram monomial is not appearing once but weighted by a weight that depends on a chord diagram invariant that is constructed from the tree $T(C)$, we call it the branch-left vector $\nu(C)$ and we will explain it in this section.

Let us consider the associated binary tree $T(C)$ to C as already defined in [7] and discussed in definition 2.2.12. For the sake of completness we explain how to obtain a binary tree $T(C)$ step by step, illustrated with a an example. First write down the decomposition of C into chord diagrams of size one as explained already in 2.2.8 and take the induced labelings on them:



Associate to each chord diagram of size one with label l a tree which contains only a root labelled l and a virtual hook. The insertion operation of trees as explained at the end of the last section associates to every chord diagram a leaf labeled binary tree $T(C)$ in this way. This tree is unique for every rooted chord diagram and what leaf labeled binary trees are possible was described in [7] and is explained in detail in Section 3.3.

Every chord corresponds to a leaf. Consider the leaf with label k. When we walk towards the root, we can count the edges passed that are heading left before heading right for the first time which we call the k-th component of the branch left vector $\nu(C)$.

This is only a very informal definition.

To make it more precise, we use a representation of a binary tree as a natural number that is inspired by a representation that is common in computer science (see [14]): Let $n = \sum_{k \geq 0} b_k 2^k$, $b_k \in \{0, 1\}$ we say that n represents a binary tree if

$$b_0 = 1 \text{ and for all } n \in \mathbb{N} : b_{2n+1} = b_{2n+2}, \text{ and } b_n = 0 \text{ must imply } b_{2n+1} = b_{2n+2} = 0.$$

We say that a vertex v is represented by k in n if the coefficient of the power 2^k in n is one, i.e. $b_k = 1$. The binary tree T is represented by $n(T)$ if

1. If v is the root of T, then v is represented by 0 in $n(T)$

2. If v is a left child of a vertex w in T that is represented by k in $n(T)$, then v is represented by $2k + 1$ in $n(T)$.

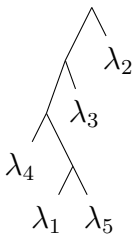
3. If v is a right child of a vertex w in T that is represented by k in $n(T)$, then v is represented by $2k+2$ in $n(T)$.

Let T be a leaf labeled binary tree and $n(T)$ the number that represents T and let $\lambda_1, \ldots, \lambda_N$ be the leaves represented by $n_1, \ldots, n_N$ in $n(T)$. Let

$$c(k) : n_k = c_1 > \ldots > c_{M(k)} = 0$$

the unique path from the leaf λ_k to the root where every c_l denotes the vertex-representation in $n(T)$ and $M(k) - 1$ the length of the path. We say that the leaf λ_k has branch-left degree ν_k if

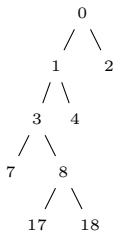
$$\nu_k(T) = \max\{N : c_l(k) \equiv 0 \,(\mathrm{mod}\ 2)\ \forall l = 1, \ldots, N\}$$

Example 2.3.1. Let $T =$ 

then the natural number that represents T is

$$n(T) = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^7 + 2^8 + 2^{17} + 2^{18} = 393631$$

which can be seen by drawing the numbers that represent the vertices in the tree:



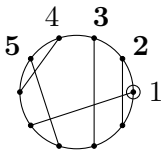
The leaves are represented as follows: λ_1 by 17, λ_2 by 2, λ_3 by 4, λ_4 by 7 and λ_5 by 18. So we have:

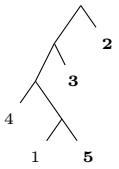
$$\begin{aligned} c(1) &: 17 > 8 > 3 > 1 > 0 \\ c(2) &: 2 > 0 \\ c(3) &: 4 > 1 > 0 \\ c(4) &: 7 > 3 > 1 > 0 \\ c(5) &: 18 > 8 > 3 > 1 > 0 \end{aligned}$$

and we see that $\nu_1 = 0, \nu_2 = 1, \nu_3 = 1, \nu_4 = 0, \nu_5 = 2$.

Definition 2.3.2 (Branch-left vector)**.** Let C be a rooted, connected chord diagram of size n and $T(C)$ its corresponding unique insertion tree, then $\nu(C) = (\nu_1, \dots, \nu_n)$ is said to be the branch-left vector of C if $\nu_k = \nu_k(T(C))$ for all $k = 1 \dots n$.

Example 2.3.3. The labeling of chords which correspond to non-zero components in the branch-left vectors are printed in bold and the branch-left vector is as calculated in the previous example.

$C =$ 

$T(C) =$ 

$\nu(C) = (0, \mathbf{1}, \mathbf{1}, 0, \mathbf{2})$

Note that the sum over all components is $n - 1$ whenever $|C| = n$. We can make this statement more precisely:

Proposition 2.3.4. *Let $\mathcal{V}(n) = \{\nu(C) = (\nu_1, \dots, \nu_n) : C \in \mathcal{R}_n\}$ the set of branch left vectors that are possible for rooted chord diagrams with size n and $\mathcal{V}(S, n) = \{\nu(C) : C \in \mathcal{R}, ter(C) = S\}$ the set of branch left vectors that are possible with fixed terminal set S. Then we have*

1. $\mathcal{V}(n) = \left\{\nu \in \mathbb{Z}^n_{\geq 0} : \sum_{k=1}^{M} \nu_k < M \; \forall M = 1 \dots n, \sum_{k=1}^{n} \nu_k = n - 1\right\}$
2. *For $S \neq \{1\}$: $\mathcal{V}(S, n) = \left\{\nu \in \mathcal{V}(n) : \nu_k \neq 0 \text{ for all } k \in S\right\}$ and for $S = \{1\}$ we have $\mathcal{V}(S, 1) = \{(0)\}$*

Proof. 1. Each $C \in \mathcal{R}_n$ decomposes into n primitive chord diagrams which gives us $n-1$ insertion operations. Each insertion operation produces exactly one additional left step on some leaf, thus $\mathcal{V}(n) \subset \{\sum_{k=1}^{n} \nu_k = n - 1\}$. It is left to prove that for $\nu = \nu(C)$ the inequalities

$$\sum_{k=1}^{M} \nu_k < M \text{ for all } M = 1, \dots, n$$

hold. But it will be noticed in Section 3.3 that $T(C)$ can be uniquely grafted by introducing a new root and putting some smaller tree H_1 on the left and a smaller tree H_2 on the right. H_1 resp. H_2 correspond uniquely (by taking the induced labeling of C) to smaller rooted connected chord diagrams D_1 resp. D_2. Since their branch left vectors do not intersect with each other, the statement follows by induction.

2. First, we notice that whenever we insert an insertion tree with some $\nu_k \neq 0$ into another insertion tree with $\nu_l \neq 0$, the resulting branch-left components are never zero, either, however exactly one component is increased by one. So, the statement follows inductively by doing the root share decomposition by continuously removing smallest removable sub trees containing 1 which are explained in Section 3.3

□

Example 2.3.5. For $n = 4$ and $S = \{3, 4\}$ the sets $\mathcal{V}(n)$, $\mathcal{V}(S, n)$ are:

$$\begin{aligned}\mathcal{V}(4) &= \Big\{\nu \in \mathbb{Z}^4_{\geq 0} : \nu_1 < 1, \nu_1 + \nu_2 < 2, \nu_1 + \nu_2 + \nu_3 < 3, \sum_{k=1}^{4} \nu_k < 4 \text{ and } \sum_{k=1}^{4} \nu_k = 3\Big\} \\ &= \Big\{(0,1,1,1), (0,0,2,1), (0,1,0,2), (0,0,1,2), (0,0,0,3)\Big\} \\ \mathcal{V}(\{3,4\}, 4) &= \Big\{(0,1,1,1), (0,0,2,1), (0,0,1,2)\Big\}\end{aligned}$$

Now, we can define the weight mentioned in the introduction of this section. Remember that it is needed for the chord diagram expansion of our Greens function.

Definition 2.3.6. [The weight of a decorated rooted connected chord diagram] For a chord diagram C with branch-left vector $\nu(C)$ and $C \in \mathcal{R}_{dec}$ with decoration $\{d_1, \ldots, d_n\}$ define

$$\omega(C) = \prod_{k=1}^{|C|} \binom{d_k s + \nu_k(C) - 2}{\nu_k(C)}$$

where $s \in \mathbb{Z}_{\geq 2}$ is the parameter given by our Dyson-Schwinger equation.

So the data we need is the terminal set and the branch left vector of a chord diagram, the next tables show how many chord diagrams there are for $|C| = 4$ and $|C| = 5$. The numbers appearing in those tables are still mysterious for us.

|C|=4	(0,1,1,1)	(0,0,2,1)	(0,1,0,2)	(0,0,1,2)	(0,0,0,3)
2 3 4	1				
2 4	1		2		
3 4	2	4		2	
4	1	2	2	4	6

C=5	1111	0211	1021	0121	0031	1102	0202	1012	0112	0022	1003	0103	0013	0004
2 3 4 5	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2 3 5	1	0	0	0	0	2	0	0	0	0	0	0	0	0
2 4 5	2	0	4	0	0	0	0	2	0	0	0	0	0	0
2 5	1	0	2	0	0	2	0	4	0	0	6	0	0	0
3 4 5	3	6	0	4	0	0	0	0	2	0	0	0	0	0
3 5	2	4	0	2	0	4	8	0	4	0	0	6	0	0
4 5	3	6	6	12	18	0	0	2	6	12	0	0	6	0
5	1	2	2	4	6	2	4	4	8	12	6	12	18	24

The entries are the number of rooted chord diagrams that correspond to the given terminal set as indicated in the row and the branch left vectors as indicated in the column. We see that the table of $|C| = 4$ is nicely embedded in the table of $|C| = 5$, which motivates the following proposition:

Proposition 2.3.7. *Let $\mathcal{R}_n(S)$ be the the rooted chord diagrams of size n with terminal set S, then there is an embedding $\iota : R_n(S) \hookrightarrow R_{n+1}(S{+}1 \cup \{2\})$ where $S{+}1 := \{s{+}1 : s \in S\}$. More precisely ι maps $(0, \nu_2, \ldots, \nu_n)$ to $(0, 1, \nu_2, \ldots, \nu_n)$*

Proof. Add a chord around the root and observe what happens in the insertion tree: A leaf with label 2 is grafted on the right of the root and all original chords except the first has its label increased by one. □

2.3.1 Branch-left intersection graph duality

Consider the set of chord diagrams of size n with given terminal set and a fixed value of the last branch-vector component:

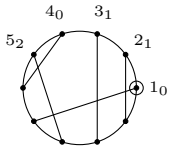
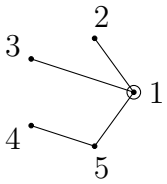
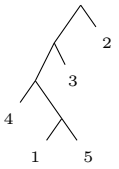
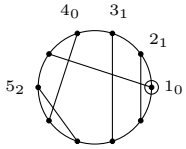
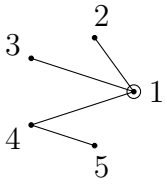
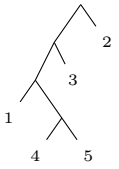
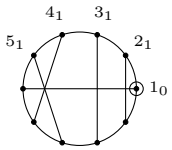
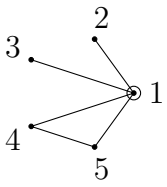
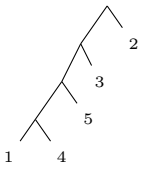
$$\mathcal{R}_n(S, m) = \{C \in \mathcal{R}_n : \mathrm{ter}(C) = S, \nu_n(C) = m\}$$

and consider the set of intersection graphs coming from chord diagrams of size n with given terminal set and a fixed number of vertices adjacent to the last vertex:

$$\mathcal{G}_n(S, m) = \{\Gamma(C) : C \in \mathcal{R}_n, \mathrm{ter}(C) = S, \delta(n) = m\}$$

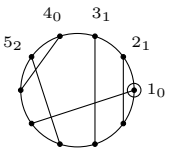
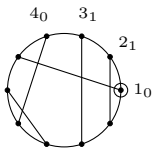
where δ denotes the vertex degree here. Then we have the conjecture that there is a bijection from $\mathcal{R}_n(S, m)$ to $\mathcal{G}_n(S, m)$ which is not the identity map $C \mapsto \Gamma(C)$. We do not know how to construct this bijection but checked that $|\mathcal{R}_n(S, m)| = |\mathcal{G}_n(S, m)|$ for every possible value of m and S for $n = 1 \ldots 8$. The following example gives a little insight of what we are talking about here:

Example 2.3.8. For $C = |5|$ with $ter(C) = \{2, 3, 5\}$ we have the following chord diagrams:

C	$\Gamma(C)$	T(C)	$\delta = (\delta_k(C))_k$	$\nu = (\nu_k(C))_k$
			(3,1,1,1,2)	(0,1,1,0,2)
			(3,1,1,2,1)	(0,1,1,0,2)
			(4,1,1,2,2)	(0,1,1,1,1)

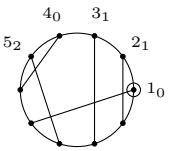
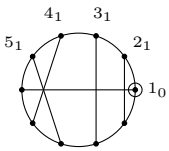
So we see that

$$\mathcal{R}_5(\{2,3,5\},2) = \left\{ \quad , \quad \right\}$$

and

$$\mathcal{G}_5(\{2,3,5\},2) = \left\{ \quad , \quad \right\}$$

In this case, only the end point of the root is changed. However, this is a special coincidence here. We do not know how this works in general. We have checked this conjecture for all chord diagram up to size eight and all corresponding terminal sets.

3 Proof of the chord diagram expansion

3.1 Overview

From now on, if nothing else is said, we consider only rooted connected chord diagrams and call them for short chord diagrams. Let $s \in \mathbb{Z}_{\geq 2}$ then the aim of this chapter is to prove that the analytic Dyson-Schwinger equation

$$G(x,L) = 1 - \sum_{k\geq 1} x^k G(x, \partial_{-\rho=0})^{1-sk}(e^{-L\rho} - 1)F_k(\rho) \text{ where } F_k(\rho) = \sum_{l\geq 0} a_{k,l}\rho^{l-1}$$

has as formal solution the combinatorial expansion in terms of chord diagrams:

$$G(x,L) = 1 - \sum_{k\geq 1} \frac{(-L)^k}{k!} \sum_{b(C)\geq k} \omega_C \hat{\underline{a}}_C a_{d_{b(C)},b(C)-k} x^{||C||}$$

In the latter equation we explain and recall the notation: $||C||$ is the sum of all decorations (a decoration is an integer d that associates a chord to some coefficient $a_{d,l}$ for some l) of C, $\underline{a}_C$ is a monomial constructed from the terminal sets that keeps track of decorations of the chord diagram, $b(C)$ is the smallest terminal chord and ω_C is the weight as defined in Definition 2.3.6. So to be more specific, let

$$ter(C) = \{t_0 < \ldots < t_n\}$$

and d_k be the decoration of the k-th chord, then we define:

$$||C|| := \sum_{c=1}^{|C|} d_c$$

$$\hat{\underline{a}}_C := \left(\prod_{c=1}^{n} a_{d_{t_c}, t_c - t_{c-1}}\right) \cdot \left(\prod_{k \notin \text{ter(C)}} a_{d_k,0}\right)$$

Here the hat does not signalize an operator but that the monomial does not contain $a_{d_{b(C)},b(C)-k}$. The symbol $b(C)$ denotes the label of the base chord and is defined as follows:

Definition 3.1.1 (Base chord b(C), $\hat{C}$ and $\omega_{\hat{C}}$)**.** For every chord diagram $C \in \mathcal{R}$, let $\mathrm{ter}(C) = \{t_0 < \ldots < t_n\}$ the set of terminal chords ordered by its labeling, then the smallest terminal chord t_0 is called the base chord and we denote it by $b(C)$. $\hat{C}$ is defined to be the chord diagram without its base chord. The weight (see 2.3.6) of $\hat{C}$ is therefore defined by

$$\omega_{\hat{C}} = \prod_{k \in C, k \neq b(C)} \binom{\nu_k + sd_k - 2}{\nu_k}$$

A simple example of the latter notations can be found at 3.2.3.

Because the theorem is not shown, yet, we want to keep track of which G we are talking about. The combinatorial expansion will be called G^{comb}:

$$G^{\mathrm{comb}}(x, L) = 1 - \sum_{k \geq 1} \frac{(-L)^k}{k!} \sum_{b(C) \geq k} \omega_C \underline{\hat{a}}_C a_{d_{b(C)},b(C)-k} x^{||C||}$$

and the $G(x, L)$ first mentioned will be therefore called G^{dif}:

$$G^{\mathrm{dif}}(x, L) = 1 - \sum_{k=0}^{N} x^k G^{\mathrm{dif}}(x, \partial_{-\rho=0})^{1-sk}(e^{-L\rho} - 1)F_k(\rho)$$
$$\text{where } G^{\mathrm{dif}}(x, L) = 1 - \sum_{k \geq 1} L^k \gamma_k(x) \text{ and } \gamma_k(x) = \sum_{l \geq k} \gamma_{k,l} x^l$$

For simplicity we define $g_k^{\mathrm{dif}}(x) = \frac{(-1)^k}{k!}\gamma_k(x)$ and analogously

$$g_k^{\mathrm{comb}}(x) = \sum_{\substack{C \in \mathcal{R}^{dec} \\ b(C) \geq k}} x^{||C||} \omega_C \underline{\hat{a}}_C a_{d_{b(C)},b(C)-k}$$

Theorem 3.1.2. *With the notation explained above, we have*

$$G^{comb}(x, L) = G^{dif}(x, L)$$

Proof. To achieve this we need to show:

1. $g_1^{\mathrm{dif}}(x) = g_1^{\mathrm{comb}}(x)$ ("the starting value is equal"):

$$g_1^{\mathrm{dif}}(x) = \sum_{C \in \mathcal{R}} x^{||C||} \omega_C \underline{\hat{a}}_C a_{d_{b(C)}, b(C)-1}$$

   This relies mainly on Theorem 3.4.13 and the renormalization group equation stated below (which is proven in the next section). The complete proof will be given after Theorem 3.4.13.

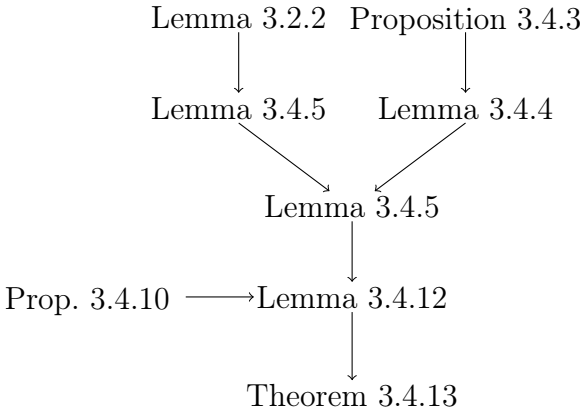
2. g^{dif} and g^{comb} follow the same recurrence (Renormalization Group Equation), i.e.

$$g_k^{\mathrm{dif/comb}}(x) = g_1^{\mathrm{dif/comb}}(x) \cdot (sx\partial_x - 1) g_{k-1}^{\mathrm{dif/comb}}(x)$$

   That g_k^{comb} satisfies the latter equation will be proven in Theorem 3.2.1. For g^{dif} this was already done in Theorem 4.10 of [16] which makes use of the Dynkin operator $S \star Y$.

□

The proof of Theorem 3.4.13 relies on the Bridge equation (Lemma 3.4.12) which requires some lemmata and propositions. The figure below illustrates the dependencies of the Bridge equations and its final result Theorem 3.4.13:



3.2 Renormalization group recurrence

The aim of this section is to prove that $g^{\mathrm{comb}}(x)$ satisfies the same recurrence as g^{dif} does. We briefly recall that:

$$g_k^{\mathrm{comb}}(x) = \sum_{\substack{C \in \mathcal{R}^{dec} \\ b(C) \geq k}} x^{||C||} \omega_C \underline{\hat{a}}_C a_{d_{b(C)}, b(C)-k}$$

and

$$g_k^{\mathrm{dif}}(x) = \frac{(-1)^k}{k!}[L^k]G^{\mathrm{dif}}(x, L)$$

Theorem 3.2.1 (Renormalization group equation for g^{comb})**.**

$$g_k^{comb}(x) = g_1^{comb}(x) \cdot (sx\partial_x - 1)g_{k-1}^{comb}(x)$$

For this theorem we need to answer the following two questions:

1. How is the monomial of a chord diagram C recovered from the root share decomposition $C = C_1 \circ_r C_2$?
2. How is the weight of a chord diagram C recovered from the root share decomposition $C = C_1 \circ_r C_2$?

The following two lemmas answer these questions and together they are enough to prove the latter theorem 3.2.1. The monomial associated to a decorated chord diagram $C \in \mathcal{R}_{\mathrm{dec}}$ with root share decomposition $C = C_1 \circ_k C_2$ can be reconstructed from C_1, C_2 in the following sense:

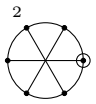
Lemma 3.2.2 (RSD monomial Lemma)**.** *Let $C_1, C_2 \in \mathcal{R}_{dec}$ with $C = C_1 \circ_k C_2$ and d, d_1, d_2 the corresponding decorations of the base chords,i.e. $d := d(b(C)), d_1 := d(b(C_1)), d_2 := d(b(C_2))$, then*

$$\underline{\hat{a}}_C a_{d,b(C)-l} = \underline{\hat{a}}_{C_1} a_{d_1,b(C_1)-1} \underline{\hat{a}}_{C_2} a_{d_2,b(C_2)-l+1}$$

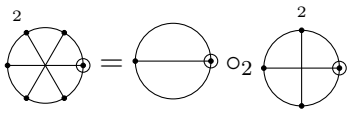
where $1 < l < b(C)$

Proof. The proof works the same as the proof of Lemma 4.2 in [7] but keeping track of decorations. □

Example 3.2.3. Let C be the wheel with three spokes as a rooted connected chord diagram and choose as decoration a two for the last chord (here the labeling is trivial and so only the non-trivial decoration is included in the pictures):



As calculated earlier $C = C_1 \circ_2 C_2$



The left hand side of the previous Lemma is:

$$\underline{\hat{a}}_C a_{d,b(C)-l} = a_{1,0}^2 a_{2,3-l}$$

The right hand side of the previous Lemma contains

$$\begin{aligned} \underline{\hat{a}}_{C_1} &= 1 \\ a_{d_1,b(C_1)-l} &= a_{1,0} \\ \underline{\hat{a}}_{C_2} &= a_{1,0} \\ a_{d_2,b(C_2)-l+1} &= a_{2,2-l+1} \end{aligned}$$

and multiplying these monomials confirms the lemma.

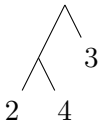
Returning to the general case, associate a weight to each branch-left vector: $\omega_C : \nu(C) \mapsto \omega_C$, to obtain an analogue of the renormalization group equation. As in the chord diagram expansion case we need to understand how the weight changes when we increase one component by one: $\omega_k : \nu \mapsto \omega + e_k$. When looking at the root share decomposition: $C = C' \circ_k C''$, we know that branch-left vector $\nu(C')$ is copied into C so only the branch left vector of C'' is modified. This yields the following equation:

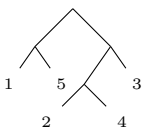
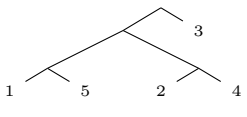
$$\sum_{k=1}^{2n-1} \omega(C' \circ_k C'') = \omega(C') \sum_{k=1}^{2n-1} \omega({}^{\circ_k}C'')$$

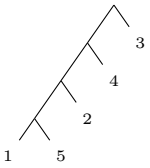
where ${}^{\circ_k}$ is defined as follows:

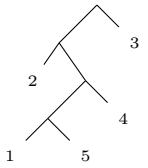
Definition 3.2.4 (Virtual insertion ${}^{\circ_k}$)**.** Let $C \in \mathcal{R}_{dec}$ then ${}^{\circ_k}C$ is defined to be the same chord diagram but with modified tree: $T({}^{\circ_k}C)$ is $T(C)$ but with an additional vertex v and an additional left child inserted before the k-th vertex w. As a result w will be the right child of v.

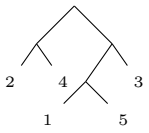
This definition seems to be very artificial. So we give an example that it will motivate, where we fix two chord diagrams and consider all possible insertions:

Example 3.2.5. Let $C' =$  and $C'' =$  and the associated trees with the induced labeling are: $T_1 := T(C') =$  and $T_2 := T(C'') =$  .

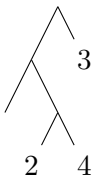
$T_1 \circ_1 T_2 =$  $T_1 \circ_2 T_2 =$ 

$T_1 \circ_3 T_2 =$ 

$T_1 \circ_4 T_2 =$ 

$T_1 \circ_5 T_2 =$ 

We note that the branch left vector of C' is never changed, so we can replace it as a marker vertex. This is what the virtual insertion does:

$T(^{\circ_2}C'') =$ 

Lemma 3.2.6. *Let C', C'' be decorated chord diagrams where $|C''| = n$, then:*

$$\sum_{k=1}^{2n-1} \omega(C' \circ_k C'') = \omega(C')\omega(C'')(s\|C''\| - 1)$$

Proof. We remember that $C' \circ_k C''$ does not affect the tree form of C' in any kind so that we get:

$$\sum_{k=1}^{2n-1} \omega(C' \circ_k C'') = \omega(C') \sum_{k=1}^{2n-1} \omega(^{\circ_k}C'')$$

Now notice that there are $\nu_k + 1$ possibilities to increase the left branch by 1.

$$\sum_{k=1}^{2n-1} \omega(^{\circ_k}C'') = (\nu_1 + 1)\omega^{+1}(C'') + \ldots + (\nu_n + 1)\omega^{+n}(C'')$$

where $\omega^{+k}(C'')$ is defined as the weight of C'' after increasing the k-th component of the branch left vector:

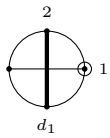
$$\omega^{+k}(C'') = \omega(C'')\left(1 + \frac{sd_k - 2}{\nu_k + 1}\right)$$

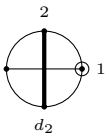
Plugging this into the latter equation, we get the result:

$$\sum_{k=1}^{2n-1}\omega(^{\circ_k}C'') = \omega(C'')\sum_{k=1}^{n}(\nu_k + 1 + sd_k - 2)$$
$$= \omega(C'')\left(n - 1 - n + s\sum_{k=1}^{n} d_k\right) = \omega(C'')(s\|C''\| - 1)$$

□

Example 3.2.7. Consider the following decorated chord diagrams (the decorated chords are thickened and the decoration is on the other side from the labeling).

$C' =$ 

$\qquad C'' =$ 

Clearly, $\nu(C') = \nu(C'') = (0, 1)$ and thus we have

$$\omega(C') = d_1 s - 1 \qquad \omega(C'') = d_2 s - 1$$
$$\omega(C')\omega(C'')(s||C''|| - 1) = (d_1 s - 1)(d_2 s - 1)(s(d_2 + 1) - 1)$$

The branch-left vectors for the different insertions are:

$$\nu(C' \circ_1 C'') = (0, 1, 0, 2) \qquad \nu(C' \circ_2 C'') = (0, 1, 1, 1) \qquad \nu(C' \circ_3 C'') = (0, 1, 0, 2)$$

Thus, for the the sum of the left hand side of the theorem we have:

$$\omega(C' \circ_1 C'') = \omega(C' \circ_3 C'') = \binom{d_2 s}{2}(d_1 s - 1)$$
$$\omega(C' \circ_2 C'') = (d_1 s - 1)(d_2 s - 1)(s - 1)$$

Summing these up, we see that l.h.s is indeed the same as the r.h.s of 3.2.6.

Proof of theorem 3.2.1. : To prove

$$g_k^{\text{comb}}(x) = g_1^{\text{comb}}(x) \cdot (sx\partial_x - 1)g_{k-1}^{\text{comb}}(x),$$

let us do the differential first:

$$(sx\partial_x - 1)g_{k-1}^{\text{comb}}(x) = \sum_{\substack{C\in\mathcal{R}^{dec} \\ b(C)\geq k-1}} (s||C|| - 1)x^{||C||}\omega_C \underline{\hat{a}}_C a_{b(C)-k+1}$$

Multiplying g_1^{comb} from the left we obtain:

$$g_1^{\mathrm{comb}}(x)\cdot(sx\partial_x-1)g_{k-1}^{\mathrm{comb}}(x)=$$
$$\left[\sum_{\substack{C'\in\mathcal{R}^{dec}\\ b(C')\geq 1}} x^{||C'||}\omega_{C'}\hat{\underline{a}}_{C'}a_{b(C')-1}\right]\left[\sum_{\substack{C''\in\mathcal{R}^{dec}\\ b(C'')\geq k-1}}(s||C''||-1)x^{||C''||}\omega_{C''}\hat{\underline{a}}_{C''}a_{b(C'')-k+1}\right]=$$
$$\sum_{\substack{C'\in\mathcal{R}^{dec},C''\in\mathcal{R}^{dec}\\ b(C')\geq 1,b(C'')\geq k-1}} x^{||C'||+||C''||}\omega_{C'}\omega_{C''}(s||C''||-1)\hat{\underline{a}}_{C'}a_{b(C')-1}\hat{\underline{a}}_{C''}a_{b(C'')-k+1}$$

By Lemma 3.2.2 and Lemma 3.2.6 the result follows. □

3.3 Leaf labels, diamonds and tree decomposition

This section explains which leaf labeled binary trees are insertion trees of chord diagrams: which shape they have and which labeling is allowed. One way to (de-)compose chord diagrams from smaller ones, that we already know, is given by the insertion operation resp. root share decomposition. This operation has a direct analogue on the level of insertion trees: the smallest removable subtree containing one, which will be defined in 3.3.7. However, there is another way of (de-)composing, which is easy to describe on the level of trees but, a priori, is not well defined on the insertion tree or chord diagram level: grafting. Let H_1 resp. H_2 be two insertion trees of rooted connected chord diagrams, say D_1 resp. D_2, then the grafting operation is $B_+(H_1H_2)$, that is the rooted plane tree given by introducing a new root and connecting H_1 to the root on the left and H_2 on the right. It is obvious that this is not a valid insertion tree since some labels appear twice. However, certain shuffles and a relabeling procedure that will be descripted in Proposition 3.3.8, yield from every H_1 and H_2 a set of grafted insertion trees. Furthermore, to every such shuffle there is a unique grafted insertion tree. This makes it possible for us to count easily the set of grafted insertion trees and will give rise to certain equations that we will need later.

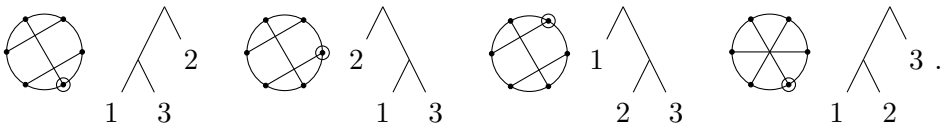
Definition 3.3.1 (Leaf label)**.** A leaf labeling for a tree T is a bijective map

$$\sigma:\Lambda(T)\to\{1,\dots,|\Lambda(T)|\}.$$

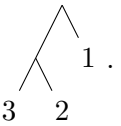
where $\Lambda(T)$ denotes the set of leaves of T.

We write (T,σ) for a leaf labeled tree T with leaf labeling σ. Not every leaf labeled tree (T,σ) is an insertion tree of a rooted chord diagram. For example, consider all rooted

chord diagrams of size 3 and their insertion trees:



We see that there is is no rooted connected chord diagram that corresponds to the tree



We call a leaf labeling that labels an insertion tree of some rooted chord diagram an admissible leaf labeling. Theorem 4.8 of [7] tells us which leaf labelings are admissible.

Proposition 3.3.2. *Let $\mathcal{T}_n$ be the set of rooted, plane, leaf labeled, binary trees with n leafs such that for every $(T, \sigma) \in \mathcal{T}_n$ the following two properties:*

P1) At any vertex v that is not a leaf the smallest label in the left subtree of v is smaller than the label at the end of the fully right branch of the right subtree.

P2) Let H be the smallest removable sub tree of T containing 1 (see Definition 4.6 of [7] or 3.3.7 for a slightly more general definition). H contains exactly the following leaf labels:

$$Im(\sigma|_H) = \Big\{1, l(T) - l(H) + 2, l(T) - l(H) + 3, \ldots, l(T)\Big\}$$

where $l(\cdot)$ denotes here the maximal label of a tree. H is the left side of the root share decomposition of trees.

Furthermore, P1 and P2 must stay true recursively in the following sense. Let $T = H \circ_r (T \setminus H)$ for some r then P1 and P2 must hold for $T \setminus H$. Then every $(T, \sigma) \in \mathcal{T}_n$ represents a unique rooted connected chord diagram of size n, so

$$\mathcal{T}_n = \{T(C) : C \in \mathcal{R}, |C| = n\}$$

Proof. See [7] □

3.3.1 Shuffling the trees of chord diagrams

To get not confused with mixing labels and label shifts in $B_+(H_1H_2)$, we introduce two copies of the natural numbers $\underline{\mathbb{N}}$ and $\overline{\mathbb{N}}$: $\underline{\mathbb{N}}$ will hold the label for the left tree and $\overline{\mathbb{N}}$ will hold the label for the right $\underline{\mathbb{N}}$ at the start of the labeling procedure. In the labeling

procedure elements of $\underline{\mathbb{N}}$ resp. $\overline{\mathbb{N}}$ will be successively replaced by the final label elements which will be elements of $\mathbb{N}$. Let $\leq$ resp. $\overline{<}$ be the strict ordering of $\underline{\mathbb{N}}$ resp. $\overline{\mathbb{N}}$. To be clear, $\leq$ resp. $\overline{<}$ is not defined to compare an element of $\underline{N}$ with an element of $\overline{N}$ and vice versa. However, there will be the point where we consider elements of $\mathbb{N}$ to be smaller than every element of a specific subset of $\underline{\mathbb{N}} \cup \overline{\mathbb{N}}$.

Let's define the shuffle product for the special case of two subsets $\{\underline{1}, \dots \underline{k}\} \subset \underline{\mathbb{N}}, \{\overline{1}, \dots, \overline{l}\} \subset \overline{\mathbb{N}}$:

$$\{\underline{1}, \dots \underline{k}\} \sqcup\!\sqcup \{\overline{1}, \dots, \overline{l}\} := \left\{ (w_1, \dots, w_{k+l}) : \begin{array}{l} \{w_1, \dots, w_{k+l}\} = \{\underline{1}, \dots \underline{k}\} \cup \{\overline{1}, \dots, \overline{l}\} \\ \text{and } r < s \Rightarrow w_r \underline{\leq} w_s \text{ if } w_r, w_s \in \underline{\mathbb{N}} \\ \text{and } r < s \Rightarrow w_r \overline{<} w_s \text{ if } w_r, w_s \in \overline{\mathbb{N}} \end{array} \right\}$$

Example 3.3.3.

$$\{\underline{1}, \underline{2}\} \sqcup\!\sqcup \{\overline{1}, \overline{2}\} = \Big\{ (\underline{1}, \underline{2}, \overline{1}, \overline{2}), (\underline{1}, \overline{1}, \underline{2}, \overline{2}), (\underline{1}, \overline{1}, \overline{2}, \underline{2}), (\overline{1}, \underline{1}, \underline{2}, \overline{2}), (\overline{1}, \underline{1}, \overline{2}, \underline{2}), (\overline{1}, \overline{2}, \underline{1}, \underline{2}) \Big\}$$

Definition 3.3.4 (Pre-labeling)**.** Let $L \subset \mathbb{N} \cup \underline{\mathbb{N}} \cup \overline{\mathbb{N}}$ a finite set. We call a bijection

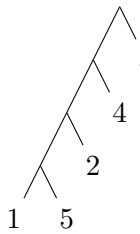
$$\sigma' : \Lambda(T) \to L$$

a pre-labeling for T if the image of σ' contains elements of $\underline{\mathbb{N}}$ or $\overline{\mathbb{N}}$.

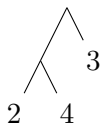
In accordance with Proposition 3.3.2, we need the smallest removable subtree containing 1 in a slightly more general setting. Before doing so, we define what we mean by removing a subtree from a tree.

Definition 3.3.5 (Removing a subtree)**.** Removing a subtree rooted at a vertex w $S \subset T$, denoted by $T \setminus S$, is defined by the following procedure:

1. Every edge and vertex from S will be removed from T.
2. The edge (w, w') where $w \in S$ and $w' \notin S$ is removed. The edge (w', w'') outgoing from w' where $w'' \notin S$ is contracted.

Example 3.3.6. Let $T =$ 

and $S =$ 

, then $T \setminus S =$ 

Definition 3.3.7 (Smallest removable subtree)**.** Let (T, σ) be a rooted, plane, leaf labeled, binary tree T with a (pre-)labeling σ. A smallest removable subtree S of T is defined to be the smallest tree such that $T \setminus S$ maintains P1 of Proposition 3.3.2.

Let $k = 1, \ldots, |D_1|$ and $m := b(D_2)$ the base chord corresponding to $D_2 = \mathcal{T}^{-1}(H_2)$, $\underline{1}, \ldots \underline{n} \in \underline{\mathbb{N}}$ the pre-labeling for H_1 and $\overline{1}, \ldots \overline{h} \in \overline{\mathbb{N}}$ the pre-labeling for H_2. The following procedure assigns to a shuffle

$$w = (w_1 \ldots w_{k+m}) \in \{\underline{1}, \ldots \underline{k}\} \sqcup\!\sqcup \{\overline{1}, \ldots, \overline{m}\}$$

an admissible labeling $\sigma = \sigma(w)$ for the grafted tree. Because this tree will be well defined, we call it $H_1 \diamond_\sigma H_2$. The set of shuffles $\{\underline{1}, \ldots \underline{k}\} \sqcup\!\sqcup \{\overline{1}, \ldots, \overline{m}\}$ is therefore the set of admissible shuffles associated to D_1, D_2 or equivalently to H_1, H_2 and will be denoted by $D_1 \sqcup\!\sqcup D_2$ resp. $H_1 \sqcup\!\sqcup H_2$.

Proposition 3.3.8. *Let w be a shuffle of $\{\underline{1}, \ldots \underline{k}\}$ and $\{\overline{1}, \ldots, \overline{m}\}$ and $(H_1, \sigma_{H_1}), (H_2, \sigma_{H_2})$ as before, then the following algorithm produces an admissible label σ and a unique leaf labeled tree $(T, \sigma) \in \mathcal{T}$:*

1. *Graft the left and right tree H_1 and H_2 at a new root, merge the pre-labelings and call this tree (T_1, σ_1). To be more specific: $T_1 = B_+(H_1 H_2)$ and*

   $$\sigma_1 : \Lambda(H_1) \cup \Lambda(H_2) \to \{\underline{1}, \ldots, \underline{n}\} \cup \{\overline{1}, \ldots, \overline{h}\}$$

   is given by

   $$\sigma_1(\lambda) = \begin{cases} \sigma_{H_1}(\lambda) & \text{if } \lambda \in H_1 \\ \sigma_{H_2}(\lambda) & \text{otherwise} \end{cases}$$

2. *For each $l = 1, \ldots, k+m$, replace the prelabel ω_l by the label $l \in \mathbb{N}$, i.e. modify σ_1 such that $\sigma_1(w_l) = l$*

3. *Assign the label $b(D_2) + k \in \mathbb{N}$ to the fully right branch leaf of T_1,i.e. modify σ_1 such that*

   $$\sigma_1(\lambda) = b(D_2) + k$$

   where λ is the leaf of the fully right branch of T_1.

4. *Apply* ***LABEL***$(T_1, \sigma_1, b(D_2)+k+1)$. *The labeling procedure* ***LABEL*** *is defined as follows:*

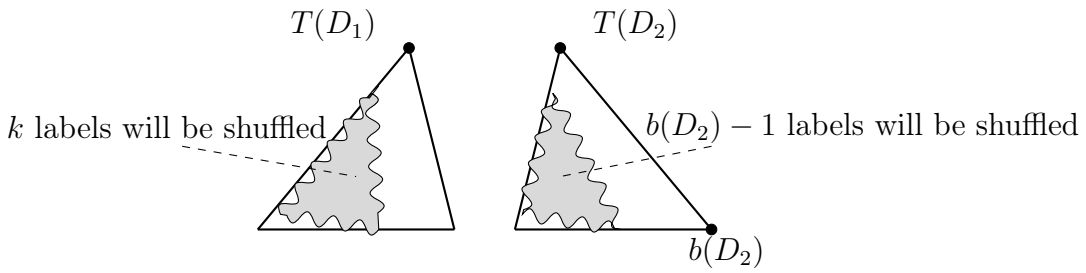
```
1    LABEL(T,ref σ,ref l) {
2      if σ is an admissible label {
3        return (T,σ)
4      }
5      s := 0
       if Im(σ) ⊂ ℕ ∪ 𝕟 or Im(σ) ⊂ ℕ ∪ ℕ̄ {
7        // replace the pre-label elements by the next labels l
8        // in the order that is induced by ≤ resp. ≼
9        if Im(σ) ⊂ ℕ ∪ 𝕟 {
10         s := |𝕟 ∩ Im(σ)|
11         Let {λ_1 ≤ ... ≤ λ_s} = 𝕟 ∩ Im(σ)
12         for i = 1...s {
13           replace pre-label λ_i by l+i
14         }
15       }
         if Im(σ) ⊂ ℕ ∪ ℕ̄ {
17         s := |ℕ̄ ∩ Im(σ)|
18         Let {λ̄_1 ≼ ... ≼ λ̄_s} = ℕ̄ ∩ Im(σ)
19         for i = 1...s {
20           replace pre-label λ̄_i by l+i
21         }
         }
       }
       // using the extended definition
25     // of smallest removable subtree to get
       // the root share decomposition on
       // the level of trees
       T = T' ∘_r T''
29     LABEL(T'',σ,l+s)
30     LABEL(T',σ,l+s)
31   }
```

Note that ***LABEL*** *does not change the form of* T.

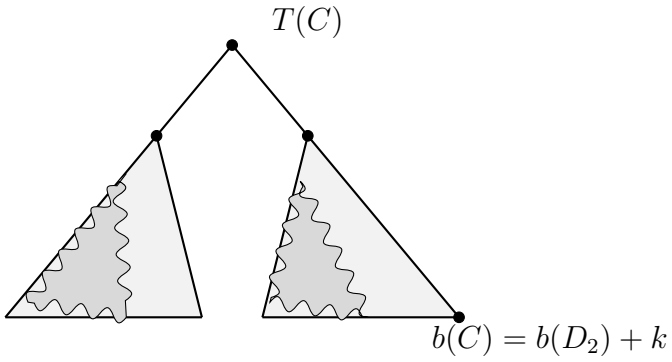
Proof. This is basically the corrected version of Lemma 4.12 of [7]. Clearly, different shuffles give different trees. The rest is labeled by the root share decomposition which is unique. □

The following schematics explain what is going on. Let us start with the trees of

$H_1 = T(D_1)$ and $H_2 = T(D_2)$:

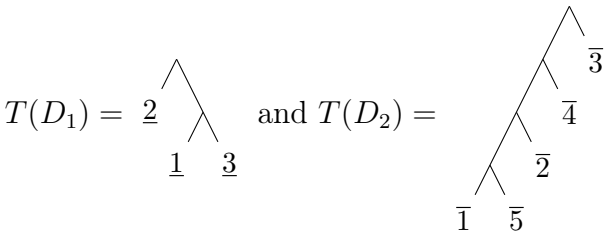


After assigning the shuffled pre-label to the grafted tree, which are $b(D_2) - 1 + k$, the next available label will be $b(D_2) + k$ which is $b(C)$



Now, we have a label one assigned to our tree, which makes it possible to look at the smallest removable sub tree containing one. We can now make the root share decomposition on trees recursively until the pre label elements of $\mathbb{N} \cup \underline{\mathbb{N}}$ are on the left side of the root share decomposition and pre label elements of $\mathbb{N} \cup \overline{\mathbb{N}}$ are on the ride side. If that is case, the pre labels will be replaced by their induced label given by the root share decomposition.

Example 3.3.9. Let us look at the following trees $T(D_1)$ and $T(D_2)$. We already replaced the labeling by the pre labeling, since this is nothing more than drawing underlines resp. overlines to the labels:



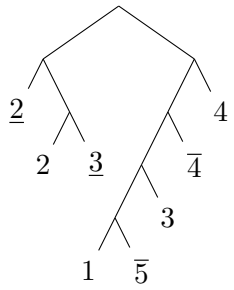
We realize immediatly that $b(D_2) = 3$. Now let us restrict to the $k = 1$ case. We need to shuffle the first $k = 1$ pre labels of $T(D_1)$ and the first $b(D_2) - 1$ labels of $T(D_2)$, so that will be:

$$\{\underline{1}\} \sqcup\!\sqcup \{\overline{1}, \overline{2}\} = \{(\underline{1}, \overline{1}, \overline{2}), (\overline{1}, \underline{1}, \overline{2}), (\overline{1}, \overline{2}, \underline{1})\}$$

Let's see how to construct the tree corresponding to the shuffle $(\overline{1}, \underline{1}, \overline{2})$. First let us assign in the tree the labels that we already know (by the steps 2 and 3 of Prop 3.3.8):

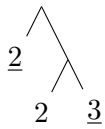
$$\overline{1} \equiv 1, \underline{1} \equiv 2, \overline{2} \equiv 3 \text{ and } \overline{3} \equiv b(D_2) + k = 4$$

and we get:



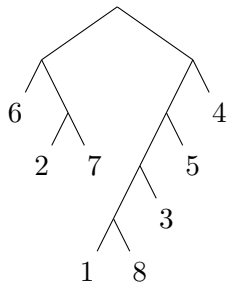
So the smallest removable subtree containing one is 

. Since the smallest removable subtree containing one is always on the left side of the root share decomposition, we know that $\overline{5}$ gets the largest label, i.e. $\overline{5} \equiv 8$. On the right side of the root share decomposition we have to look at what is the smallest removable subtree containing two (we would not allow the smallest label of a tree to be two, but we are working with the induced labeling which makes it easier to calculate and not to get confused). The smallest removable subtree containing two is then:

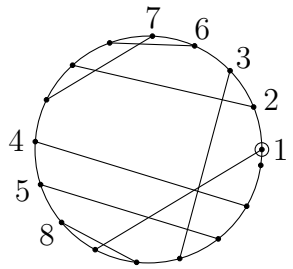


We see that this root share decomposition splits elements of $\underline{\mathbb{N}}$ and $\overline{\mathbb{N}}$, so we can resolve:

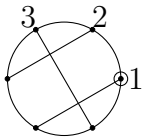
$\underline{2} \equiv 6, \underline{3} \equiv 7, \overline{4} \equiv 5$. So the final tree is:

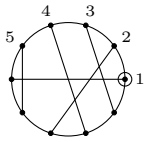


which corresponds to:



where

$D_1 =$ 

and $D_2 =$ 

3.4 Bridge equation

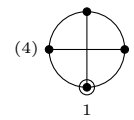
As mentioned previously, the root share decomposition is not the only way to decompose chord diagrams into smaller ones. When starting with a tree $T(C)$ associated to a chord diagram, by removing the root we obtain a left and a right tree that define chord diagrams by themselves. Let us call those chord diagrams D_1 and D_2, They are well defined for every chord diagram C and so we are able to define the diamond operation on chord diagrams. This operation, which will be defined in detail in 3.4.1, is needed for some technical lemmas that we need to prove the main theorem. It roughly says that summing over a set of chord diagrams of fixed size n is the same as summing over all possible decompositions of C into D_1, D_2. To be more specific we will need to prove:

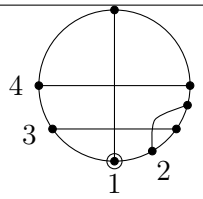
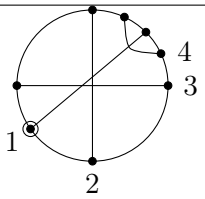
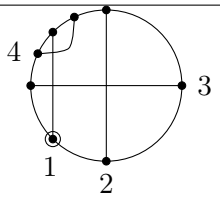
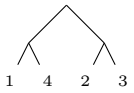
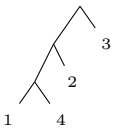
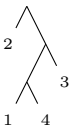
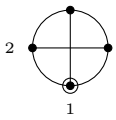
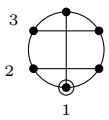
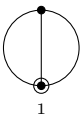
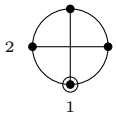
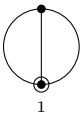
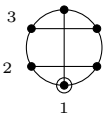
$$\sum_{\substack{||C||=i+1 \\ b(C)=j+1}} \omega_{\hat{C}} \underline{\hat{a}}_C = \sum_{k=1}^{i} \sum_{l=1}^{j} \binom{j}{l} \left(\sum_{\substack{||D_1||=k \\ b(D_1)\geq l}} \omega_{D_1} \underline{\hat{a}}_{D_1} a_{b(D_1)-l} \right) \left(\sum_{\substack{||D_2||=i-k+1 \\ b(D_2)=j-l+1}} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_2} \right) \tag{3.4.1}$$

The proof relies crucially on the Proposition 3.3.8, which tells us in how many ways two given trees can be grafted together.

If we decompose trees or chord diagrams by their left and right subtree, this is a well defined operation. However, if we start with two trees it is not clear which labeling the diamond operation should put out and Proposition 3.3.8 tells us what labelings are possible for it. This being said, we define the diamond operation in the following way

Definition 3.4.1 (Diamond operation on trees and chord diagrams)**.** Let $T_1, T_2 \in T(\mathcal{R})$, λ be a leaf labeling of size $l(T_1) + l(T_2)$ where $l(\cdot)$ denotes the numbers of leafs, then we define $T_1 \diamond_\lambda T_2$ to be the unique tree that has T_1 as left tree, T_2 as right tree and λ as leaf labeling. If we take the induced labeling of a tree T, we write $T_1 \diamond_T T_2$. Analogously, we write for chord diagrams D_1, D_2 and a chord labeling μ of size $|D_1| + |D_2|$ $D_1 \diamond_\mu D_2$ and for the induced labeling of a chord diagram C, we write $D_1 \diamond_C D_2$. If no labeling is assigned, i.e. $T_1 \diamond T_2$ resp. $D_1 \diamond D_2$ is defined to be the set of all possible labelings.

Example 3.4.2. Consider $C =$  Then depending on k we have the following C and $C = D_1 \diamond_C D_2$

k	1	2	3
C			
$T = H_1 \diamond_T H_2$			
H_1			
H_2			
D_1			
D_2			

The diamond operation is compatible with the root share decomposition, however, it is not always combatible the same way. It depends on the insertion place of the root share decomposition as stated below:

Proposition 3.4.3. *Let C be a chord diagram with $|C| \geq 3$ and $C = C' \circ_k C_2$ where*

$C_2 = C'' \diamond_{C_2} C'''$, *then*

$$C' \circ_k (C'' \diamond_{C_2} C''') = \begin{cases} (C' \circ_{k-1} C'') \diamond_C C''' & |C''| \leq k-1, k > 1 \\ C' \diamond_C (C'' \diamond_C C''') & k = 1 \\ C'' \diamond_C (C' \circ_{k-|C''|-1} C''') & else \end{cases}$$

Proof. Let $|C| \geq 3$ with $C = C' \circ_k C_2$ then there are three cases to consider if we look at $T = T(C)$:

1. k is the root of T: This is the case $k = 1$ and so root share decomposition and diamond decomposition coincidence.

2. k lies in the left subtree of T: Let $T(D_1)$ be the left subtree of T and D_1' the diagram coresponding chord diagram to left subtree of $T(D_1)$, then

$$D_1 = C' \circ_{k-1} D_1'$$

3. k lies in the right subtree of T: Let $T(D_2)$ be the right subtree of T and D_2' the diagram coresponding chord diagram to left subtree of $T(D_2)$, then

$$D_2 = C' \circ_{k-1-|D_1|} D_2'$$

□

Actually, there is a "triangle inequality" for base chords, which is Lemma 4.10 in [7] and which we will show by using the former "associativity law". It is required for Proposition 3.4.8.

Lemma 3.4.4 (Triangle inequality for the base chords)**.**

$$b(D_1 \diamond D_2) \leq b(D_1) + b(D_2)$$

Proof. Induction on number of chords of $D_1 \diamond D_2$: Induction start is trivial. So there are three different cases to consider:

1. $D_1 \diamond D_2 = (D_1' \circ_{k-1} D_1'') \diamond D_2 = D_1' \circ_k (D_1'' \diamond D_2)$ for some k:

$$\begin{aligned} b(D_1 \diamond D_2) &= b(D_1'' \diamond D_2) + 1 \\ &\leq b(D_1'') + b(D_2) + 1 \\ &= b(D_1) - 1 + b(D_2) + 1 \end{aligned}$$

2. $b(D_1 \diamond D_2) = b(D_1 \circ_1 D_2) = b(D_2) + 1 \leq b(D_2) + b(D_1)$

3. $D_1 \diamond (D_2' \circ_{k-|D_2'|-1} D_2'') = D_2' \circ_k (D_1 \diamond D_2'')$:

$$\begin{aligned} b(D_1 \diamond D_2) &= b(D_1 \diamond D_2'') \\ &\leq b(D_1) + b(D_2) - 1 < b(D_1) + b(D_2) \end{aligned}$$

□

The terminal sets are under control for $\circ_k$ but we don't know what they do for $\diamond$. The branch-left vectors are under control for $\diamond$ (let $C = D_1 \diamond D_2$, then the only component that is increased is the base chord of D_2) but we know only partial results on $\circ_k$. To prove the equation 3.4.1 we need the following Lemma which uses the last lemma and compares the diamond and insertion operation. It uses the same principle as we needed in the proof of the triangle inequality: three cases depending on the insertion place.

Lemma 3.4.5. *Let $C \in \mathcal{R}$ with $|C| \geq 2$ and T, H_1, H_2, D_1, D_2 as before. Let $d = d_{b(D_1)}$ be the decoration of the smallest terminal set of D_1, then*

$$\omega_{\hat{C}} \underline{\hat{a}}_C = \omega_{D_1} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_1} \underline{\hat{a}}_{D_2} a_{d,b(D_1)+b(D_2)-b(C)}$$

Proof. The Lemma follows from the following two claims:

Claim 1: $\omega_{\hat{C}} = \omega_{D_1} \omega_{\hat{D}_2}$. Remember that $\omega_{\hat{C}} = \prod_{k \neq b(C)} \binom{sd_k + \nu_k - 2}{\nu_k}$ and notice that in the induced labeling the base chord of D_2 and C are the same, so we have $b(D_2) = b(C)$ because it is the fully right branch leaf of the tree that correspond to D_2 as well that of C.

Claim 2: $\underline{\hat{a}}_C = \underline{\hat{a}}_{D_1} \underline{\hat{a}}_{D_2} a_{d,b(D_1)+b(D_2)-b(C)}$. This claim can be proved analogously to 4.11 out of [7]. For the sake of completeness we carry it out: The main point to study here is to compare the root share decomposition $C = C_1 \circ_k C_2$ with the tree decomposition $C = D_1 \diamond D_2$. There are three cases to investigate:

a) $k = 1$: We note that

$$d_{b(C)} = d_{b(D_2)} = d_{b(C_2)} \text{ and } b(C_2) = b(D_2) = b(C) - 1$$

because Root share decomposition and tree decomposition coincidence. This being said, we can use directly Lemma 3.2.2 now:

$$\begin{aligned} \underline{\hat{a}}_C a_{d_{b(D_2)}, b(D_2)-l+1} &= \underline{\hat{a}}_{D_1} a_{d_{b(D_1)}, b(D_1)-1} \underline{\hat{a}}_{D_2} a_{d_{b(D_2)}, b(D_2)-l+1} \\ &= \underline{\hat{a}}_{D_1} a_{d_{b(D_1)}, b(D_1)+b(D_2)-b(C)} \underline{\hat{a}}_{D_2} a_{d_{b(D_2)}, b(D_2)-l+1} \end{aligned}$$

b) k is an edge in the left tree $T(D_1) = H_1$: First, by induction the statement holds for C_2. Thus let D_1', D_2' the chord diagrams corresponding to the left and the right tree of $T_2 = T(C_2)$, then

$$\underline{\hat{a}}_{C_2} = \underline{\hat{a}}_{D_1'} \underline{\hat{a}}_{D_2'} a_{d(b(C)),b(D_1')+b(D_2')-b(C_2)} \tag{3.4.2}$$

Second, apply Lemma 3.2.2 on $D_1 = C_1 \circ_{k-1} D_1'$ with $l = k$ to obtain

$$\underline{\hat{a}}_{D_1} a_{d(b(D_1)),b(D_1)-(k-1)} = \underline{\hat{a}}_{C_1} a_{d(b(C_1)),b(C_1)-1} \underline{\hat{a}}_{D_1'} a_{d(b(D_1')),b(D_1')-k+2} \tag{3.4.3}$$

And we have by applying Lemma 3.2.2 on $C = C_1 \circ_k C_2$:

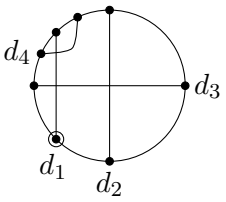
$$\begin{aligned}
\underline{\hat{a}}_C a_{d(b(C)),b(C)-(k-1)} &= \underline{\hat{a}}_{C_1} a_{d(b(C_1)),b(C_1)-1} \underline{\hat{a}}_{C_2} a_{d(b(C_2)),b(C_2)-k+1} \\
&\overset{3.4.3}{=} \frac{\underline{\hat{a}}_{D_1} a_{d(b(D_1)),b(D_1)-(k-1)}}{\underline{\hat{a}}_{D_1'} a_{d(b(D_1')),b(D_1')-k+2}} \underline{\hat{a}}_{C_2} a_{d(b(C_2)),b(C_2)-k+1} \\
&\overset{3.4.2}{=} \frac{\underline{\hat{a}}_{D_1} a_{d(b(D_1)),b(D_1)-(k-1)}}{\underline{\hat{a}}_{D_1'} a_{d(b(D_1')),b(D_1')-k+2}} \\
&\quad \times \underline{\hat{a}}_{D_1'} \underline{\hat{a}}_{D_2'} a_{d(b(C)),b(D_1')+b(D_2')-b(C_2)} a_{d(b(C_2)),b(C_2)-k+1}
\end{aligned}$$

The result follows with $b(C) = b(C_2) + 1$ and $b(D_1) = b(D_1') + 1$ and because $D_2' = D_2$ remains unchanged.

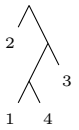
c) k is an edge in the right tree $T(D_2) = H_2$: Then $D_2 = C_1 \circ_{k-1-|D_1|} D_2'$ and the calculation works analogously as the last case and the results follows with the help of $b(C) = b(C_2) + 1$ and $b(D_2) = b(D_2') + 1$.

□

Example 3.4.6. Consider the following chord diagram with arbitrary decorations $d_1, \ldots, d_4$ and arbitrary $s \neq 1$:

$C =$ 

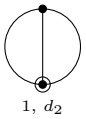
It has terminals $\mathrm{ter}(C) = \{3, 4\}$, so $b(C) = 3$. The corresponding tree is

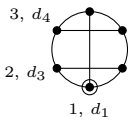
$T(C) =$ 

so all in all we have for the left hand side of the previous lemma:

$$\omega_{\hat{C}} \underline{\hat{a}}_C = (d_4 s - 1) a_{d_4,1} a_{d_1,0} a_{d_2,0}$$

For the right hand side we have the following trees and diagrams (the decoration is inherited but the labeling is normalized):

$D_1 =$ 

$D_2 =$ 

$H_1 =$ 

$H_2 =$ 

So we have for the right hand side:

$$\begin{aligned}
&\underline{\hat{a}}_{D_1} = 1 \\
&\underline{\hat{a}}_{D_2} = a_{d_4,1} a_{d_1,0} \\
&\omega_{D_1} = 1 \\
&\omega_{\hat{D}_2} = d_4 s - 1 \\
&a_{d(b(D_1)),b(D_1)+b(D_2)-b(C)} = a_{d_2,0}
\end{aligned}$$

multiplying this we indeed get the same as the left hand side as stated by the Lemma.

Proposition 3.4.7. *We first have the following observation for $j \in \mathbb{Z}_{\geq 0}$ and every $k \in \mathbb{N}$*

$$\sum_{\substack{||C||=i+1 \\ d_{j+1}=1 \\ \nu_{j+1}=n \\ b(C)=j+1}} \underline{\hat{a}}_C \omega_{\hat{C}} = \sum_{\substack{||C||=i+k \\ d_{j+1}=k \\ \nu_{j+1}=n \\ b(C)=j+1}} \underline{\hat{a}}_C \omega_{\hat{C}}$$

Proof. Let

$$\mathcal{C}_{i,j,k} := \left\{ C \in \mathcal{R}^{\mathrm{dec}} : ||C|| = i + k, d_{j+1} = k, \nu_{j+1} = n, b(C) = j + 1 \right\}$$

We need to show that

$$A_{i,j} := \{ \underline{\hat{a}}_C \omega_{\hat{C}} : C \in \mathcal{C}_{i,j,1} \} \text{ is bijective to every } A_{i,j,k} := \{ \underline{\hat{a}}_C \omega_{\hat{C}} : C \in \mathcal{C}_{i,j,k} \}$$

For a fixed k the map that replaces the decoration $d_{j+1} = 1$ by k clearly defines a bijection between $\mathcal{C}_{i,j,1}$ and $\mathcal{C}_{i,j,k}$. This map lifts to a bijection $A_{i,j} \leftrightarrow A_{i,j,k}$ because the decoration d_{j+1} is ignored by definition of $\underline{\hat{a}}_C$ and $\omega_{\hat{C}}$, since $b(C) = j + 1$. □

Proposition 3.4.8 (Decorated version of Prop 4.3 from [7])**.**

$$\sum_{\substack{||C||=i+1\\ b(C)=j+1}} \omega_{\hat{C}} \underline{\hat{a}}_C = \sum_{k=1}^{i} \sum_{l=1}^{j} \binom{j}{l} \left(\sum_{\substack{||D_1||=k\\ b(D_1)\geq l}} \omega_{D_1} \underline{\hat{a}}_{D_1} a_{b(D_1)-l} \right) \left(\sum_{\substack{||D_2||=i-k+1\\ b(D_2)=j-l+1}} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_2} \right)$$

Proof. We know that each chord diagram C of size $i + 1$ can be tree decomposed to $C = D_1 \diamond D_2$ and that in this case $b(D_1) + b(D_2) \geq b(C)$ by the triangle inequality 3.4.4. However, given $b(D_1) \geq l$ for fixed l and $b(D_2) = j - l + 1$ for fixed j results in $\binom{j}{l}$ possibilites for D_1, D_2 such that $C = D_1 \diamond D_2$ by Proposition 3.3.8. Furthermore $b(C) = j + 1$. Therefore the sum on the left hand side of the statement splits like:

$$\sum_{\substack{||C||=i+1\\ b(C)=j+1}} = \sum_{k=1}^{i} \sum_{l=1}^{j} \binom{j}{l} \left(\sum_{\substack{||D_1||=k\\ b(D_1)\geq l}} \right) \left(\sum_{\substack{||D_2||=i-k+1\\ b(D_2)=j-l+1}} \right)$$

Now given a monomial $\omega_{\hat{C}} \underline{\hat{a}}_C$ we know how to decompose it into the monomials as needed by Lemma 3.4.5. Inserting them into the sums proves the proposition. □

Example 3.4.9. Let $i = 3, j = 1$ and $N \geq 3$ and $s \in \mathbb{Z}_{\geq 2}$ arbitrary. We have to consider all chord diagrams with $||C|| = 4, b(C) = 2$. For the decorations we need to consider all compositions of four:

$$(1,1,1,1), (2,1,1), (1,2,1), (1,1,2), (2,2), (1,3), (3,1)$$

Because of the constraint $b(C) = 2$ all chord diagrams to consider are

$$\mathcal{R}_{\{2,3,4\}}, \mathcal{R}_{\{2,4\}}, \mathcal{R}_{\{2,3\}}, \mathcal{R}_{\{2\}}$$

For $\mathcal{R}_{\{2,3,4\}}$ and $\mathcal{R}_{\{2,4\}}$ only the decoration $d_1 = d_2 = d_3 = d_4 = 1$ is possible so we calculate these: We have one chord diagram with $ter = \{2, 3, 4\}$ and branch left vector $(0, 1, 1, 1)$:

$$\underline{\hat{a}}_C = a_1^2 a_0 \text{ and } \omega_{\hat{C}} = (s-1)^2$$

There are three chord diagrams with $ter = \{2, 4\}$, namely the two chord diagrams with

branch left vector $(0,1,0,2)$:

$$\underline{\hat{a}}_C = a_2 a_0^2 \text{ and } \omega_{\hat{C}} = \binom{s}{2}$$

and the one with branch left vector $(0,1,1,1)$:

$$\underline{\hat{a}}_C = a_2 a_0^2 \text{ and } \omega_{\hat{C}} = (s-1)^2$$

Summing this up $\mathcal{R}_{\{2,3,4\}}$ and $\mathcal{R}_{\{2,4\}}$ contribute to the left hand side with:

$$(s-1)^2 a_1^2 a_0 + a_2 a_0^2 \left(2\binom{s}{2} + (s-1)^2\right) = (s-1)^2 a_1^2 a_0 + (2s^2 - 3s + 1) a_2 a_0^2$$

$\mathcal{R}_{\{2,3\}}$ consists only of one chord diagram with branch left vector $(0,1,1)$ and we have to consider the decorations: $(d_1, d_2, d_3) \in \{(2,1,1),(1,2,1),(1,1,2)\}$, so it contributes on the left hand side with:

$$\left((2s-1) + 2(s-1)\right) a_0 a_1 = (4s-3) a_0 a_1$$

$\mathcal{R}_{\{2\}}$ has only one chord diagram which contributes to left hand side by $3a_0$. The weight is 1 because the branch left vector of the chord diagram is $(0,1)$ where the second coordinate is ignored by $\omega_{\hat{C}}$, but there are three chord diagrams to consider, namely that are decorated by $(1,3),(3,1),(2,2)$. So the left hand side of last lemma is for this example:

$$(s-1)^2 a_1^2 a_0 + (2s^2 - 3s + 1) a_2 a_0^2 + (4s-3) a_0 a_1 + 3a_0$$

For the RHS we need to take the induced labels of D_1 and D_2 and we have to consider only the last sum:

$$\begin{aligned}
\text{RHS} = &\left[\sum_{\substack{||D_1||=1 \\ b(D_1)\geq 1}} \omega_{D_1} \underline{\hat{a}}_{D_1} a_{b(D_1)-1}\right] \cdot \left[\sum_{\substack{||D_2||=3 \\ b(D_2)=1}} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_2}\right] \\
&+ \left[\sum_{\substack{||D_1||=2 \\ b(D_1)\geq 1}} \omega_{D_1} \underline{\hat{a}}_{D_1} a_{b(D_1)-1}\right] \cdot \left[\sum_{\substack{||D_2||=2 \\ b(D_2)=1}} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_2}\right] \\
&+ \left[\sum_{\substack{||D_1||=3 \\ b(D_1)\geq 1}} \omega_{D_1} \underline{\hat{a}}_{D_1} a_{b(D_1)-1}\right] \cdot \left[\sum_{\substack{||D_2||=1 \\ b(D_2)=1}} \omega_{\hat{D}_2} \underline{\hat{a}}_{D_2}\right]
\end{aligned}$$

Since $N \in \mathbb{N}_{\geq 3}$, we have

$$\sum_{\substack{||D_2||=3\\ b(D_2)=1}} \omega_{\hat{D}_2} \hat{\underline{a}}_{D_2} = \sum_{\substack{||D_2||=2\\ b(D_2)=1}} \omega_{\hat{D}_2} \hat{\underline{a}}_{D_2} = \sum_{\substack{||D_2||=1\\ b(D_2)=1}} \omega_{\hat{D}_2} \hat{\underline{a}}_{D_2} = 1$$

$$\sum_{\substack{||D_1||=1\\ b(D_1)\geq 1}} \omega_{D_1} \hat{\underline{a}}_{D_1} a_{b(D_1)-1} \cdot = a_0$$

$$\sum_{\substack{||D_1||=2\\ b(D_1)\geq 1}} \omega_{D_1} \hat{\underline{a}}_{D_1} a_{b(D_1)-1} = (s-1) a_0 a_1 + a_0$$

$$\sum_{\substack{||D_1||=3\\ b(D_1)\geq 1}} \omega_{D_1} \hat{\underline{a}}_{D_1} a_{b(D_1)-1} = (s-1)^2 a_0 a_1^2 + (2s^3 - 3s + 1) a_0^2 a_2 + ((s-1) + (2s-1))\, a_0 a_1 + a_0$$

Summing this up, we get indeed the left hand side.

Proposition 3.4.10 (Restricted decorated Prop 4.3)**.**

$$\sum_{\substack{||C||=i+1\\ b(C)=j+1\\ \nu_{b(C)}=n}} \omega_{\hat{C}} \hat{\underline{a}}_C = \sum_{k=1}^{i} \sum_{l=1}^{j} \binom{j}{l} \left(\sum_{\substack{||D_1||=k\\ b(D_1)\geq l}} \omega_{D_1} \hat{\underline{a}}_{D_1} a_{b(D_1)-l} \right) \left(\sum_{\substack{||D_2||=i-k+1\\ b(D_2)=j-l+1\\ \nu_{b(D_2)}=n-1}} \omega_{\hat{D}_2} \hat{\underline{a}}_{D_2} \right)$$

Proof. Note that $\nu_{b(D_1 \diamond D_2)}(D_1 \diamond D_2) = \nu_{b(D_2)}(D_2) + 1$. Indeed, $b(D_1 \diamond D_2) = b(D_2)$ and attaching a tree on the left side of $T(D_2)$ increments the right most branch, which ends at the leaf $b(D_2)$. Therefore the length of the rightmost branch matches as given in the proposition. The rest follows by Proposition 3.4.8 □

In the following we develop equations that builds a bridge between the sum that contains only chord diagrams and the sum that is calculated by derivations. Therefore, we will call it the Bridge equation. It is proved by induction and the following lemma makes the start for the induction whereas the lemma after it states the general case.

Lemma 3.4.11. *Let $i \geq 1, j \geq 1$, then*

$$\sum_{\substack{||C||=i+1 \\ d_{j+1}=1 \\ \nu_{j+1}=1 \\ b(C)=j+1}} \underline{\hat{a}}_C \omega_{\hat{C}} = [x^i] \left(\sum_{l \geq 1} \frac{g_l^{comb}(x)}{l!} \partial_{\rho=0}^l \right) \rho^j$$

Proof. Because the right hand side of the equation has only one non-zero term, namely the term where $l = j$, therefore it suffices to prove

$$\sum_{\substack{||C||=i+1 \\ d_{j+1}=1 \\ \nu_{j+1}=1 \\ b(C)=j+1}} \underline{\hat{a}}_C \omega_{\hat{C}} = \sum_{\substack{||D||=i \\ b(D) \geq j}} \omega_D \underline{\hat{a}}_D a_{d(b(D)),b(D)-j} \tag{3.4.4}$$

We know that for $C = D \diamond D_2$ with D_2 being the chord diagram with only one chord diagram. Therefore we have the triangle inequality for the label of the base chords hold: $b(D) \geq b(C) - b(D_2)$. We note that there indeed exists only one C with this diamond decomposition and $b(C) = j + 1$, because there is only one shuffle which ends with the integer $j + 1$. Now, on the sum of the LHS every chord diagram splits into $D \diamond D_2$ with D_2 being the chord diagram with only one chord. But D now has the size $||D|| = ||C|| - d_{j+1} = i$ and its base chord satisfies $b(D) \geq j + 1 - 1 = j$. Thus it is left to see that the corresponding summands are actually equal on both sides of Eq. 3.4.4. But this was already done in Lemma 3.4.5 □

Now, we can use it to prove the statement that we were looking for:

Lemma 3.4.12 ("Bridge equation")**.** *Let $n \geq 1$, then*

$$\sum_{\substack{||C||=i+1 \\ d_{j+1}=1 \\ \nu_{j+1}=n \\ b(C)=j+1}} \underline{\hat{a}}_C \omega_{\hat{C}} = [x^i] \left(\sum_{l \geq 1} \frac{g_l^{comb}(x)}{l!} \partial_{\rho=0}^l \right)^n \rho^j$$

Proof. For better readability define: $G_{\partial\rho}(x) := \sum_{l \geq 1} \frac{g_l^{\text{comb}}(x)}{l!} \partial_{\rho=0}^l$ and $F_{i,j,n} := [x^i] G_{\partial\rho}^n(x) \rho^j$. Let i, j be fixed, we prove the statement by induction over n. For $n = 1$ the statement is true by the previous Lemma. Now analogously to the proof of Lemma 4.14 in [7], we observe that:

$$F_{i,j,n} = [x^i] G_{\partial\rho}^n(x) \rho^j$$

$$
\begin{aligned}
&= \sum_{k=1}^{i} \left([x^k]G_{\partial\rho}(x)\right)\left([x^{i-k}]G_{\partial\rho}^{n-1}(x)\right)\rho^j \\
&\overset{\text{Leibniz-Rule}}{=} \sum_{k=1}^{i}\sum_{l=1}^{j}\binom{j}{l}\left([x^k]G_{\partial\rho}(x)\rho^l\right)\left([x^{i-k}]G_{\partial\rho}^{n-1}\rho^{j-l}\right) \\
&\overset{\text{Definition}}{=} \sum_{k=1}^{i}\sum_{l=1}^{j}\binom{j}{l}[x^k]g_l^{\text{comb}}(x)\cdot F_{i-k,j-l,n-1} \\
&\overset{\text{induction}}{=} \sum_{k=1}^{i}\sum_{l=1}^{j}\binom{j}{l}\left(\sum_{\substack{||D_1||=k\\ b(D_1)\geq l}}\omega_{D_1}\hat{\underline{a}}_{D_1}a_{b(D_1)-l}\right)\left(\sum_{\substack{||D_2||=i-k+1\\ b(D_2)=j-l+1\\ \nu_{b(D_2)}=n-1}}\omega_{\hat{D}_2}\hat{\underline{a}}_{D_2}\right) \\
&\overset{\text{Prop 3.4.10}}{=} \sum_{\substack{||C||=i+1\\ d_{j+1}=1\\ \nu_{j+1}=n}}\hat{\underline{a}}_C\omega_{\hat{C}}
\end{aligned}
$$

□

Theorem 3.4.13. *Let* $G_{\partial\rho}^{comb} := \sum_{l\geq 1}\frac{g_l^{comb}}{l!}$ *and* $\tilde{F}_k(\rho) := \sum_{l\geq 0}a_{k,l}\rho^l$*, then*

$$g_1^{comb} = \sum_{k=1}^{N} x^k \sum_{n\geq 0}\binom{n+sk-2}{n}(G_{\partial\rho}^{comb})^n\tilde{F}_k(\rho)$$

Proof. For the first coefficient of the right hand side of the equation that we want to show, we have:

$$[x^i]\text{RHS} = \sum_{k=1}^{N}\sum_{n\geq 0}\binom{n+sk-2}{n}\sum_{l\geq 0}a_{k,l}[x^{i-k}](G_{\partial\rho}^{\text{comb}})^n\rho^l$$

Now, we can use the "Bridge Equation" and 3.4.7:

$$\sum_{\substack{||C||=i+k\\ d_{l+1}=k\\ \nu_{l+1}=n\\ b(C)=l+1}}\hat{\underline{a}}_C\omega_{\hat{C}} = [x^i](G_{\partial\rho}^{\text{comb}})^n\rho^l \quad \text{for all } k = 1\ldots N.$$

And we get:

$$
\begin{aligned}
[x^i]\mathrm{RHS} &= \sum_{k=1}^{N}\sum_{n\geq 0}\binom{n+sk-2}{n}\sum_{l\geq 0} a_{k,l}\sum_{\substack{||C||=i\\ d_{l+1}=k\\ \nu_{l+1}=n\\ b(C)=l+1}} \underline{\hat{a}}_C\omega_{\hat{C}} \\
&= \sum_{k=1}^{N}\sum_{n\geq 0}\sum_{l\geq 0} a_{k,l}\sum_{\substack{||C||=i\\ d_{l+1}=k\\ \nu_{l+1}=n\\ b(C)=l+1}} \underline{\hat{a}}_C\omega_{\hat{C}}\binom{\nu_{l+1}+sk-2}{\nu_{l+1}} \\
&= \sum_{k=1}^{N}\sum_{n\geq 0}\sum_{l\geq 0}\sum_{\substack{||C||=i\\ d_{b(C)}=k\\ \nu_{b(C)}=n\\ b(C)=l+1}} a_{k,b(C)-1}\underline{\hat{a}}_C\omega_{\hat{C}}\binom{\nu_{l+1}+sk-2}{\nu_{l+1}}
\end{aligned}
$$

Since we have $l = b(C) + 1$, we realize that

$$\omega_{\hat{C}}\binom{\nu_{l+1}+sk-2}{\nu_{l+1}} = \omega_C$$

We look carefully at the restrictions of the last sum. We need to verify that we can drop the last three constraints because we are summing over all possible k, n, l:

1. The restriction $d_{b(C)} = k$ drops because we are summing over all $k = 1 \dots N$.
2. The restriction of $\mathcal{R}_{dec}$ to $||C|| = i, \nu_{b(C)} = n$ is always non-empty for some n and summing over all n yields indeed all rooted connected decorated chord diagrams with $||C|| = i$.
3. $b(C) \geq 1$ so we can drop $b(C) = l + 1$ and $\sum_l$

In conclusion:

$$[x^i]\mathrm{RHS} = \sum_{||C||=i}\omega_C\underline{\hat{a}}_C a_{d(b(C)),b(C)-1} = g_1^{\mathrm{comb}}$$

□

Theorem 3.1.2 1. Since the Renormalization Group equation holds true for both g^{comb} (by Theorem 3.2.1) and g^{dif} (by Theorem 4.2 of [16]) , $G^{\mathrm{dif/comb}}$ are build from $g_1^{\mathrm{dif/com}}$ completely in the same way:

$$g_k^{\mathrm{dif/comb}} = g_1^{\mathrm{dif/comb}}(sx\partial_x - 1)g_{k-1}^{\mathrm{dif/comb}} \text{ for } k \geq 2$$

For g_1^{dif} we have by applying the generalized geometric series to $G^{dif}(x, \partial_{-\rho})^{1-sk}$:

$$g_1^{\text{dif}} = \sum_{k=1}^{N} x^k \sum_{n\geq 0} \binom{n+sk-2}{n} (G_{\partial\rho}^{\text{dif}})^n \tilde{F}_k(\rho)$$

where, as mentioned in the introduction, $\tilde{F}_k(\rho) = (e^{-L\rho}-1)\frac{1}{\rho}\sum_{l\geq 0} a_{k,l}\rho^l$. The last theorem 3.4.13 helps us to show equality of $g_1^{\text{comb}} = g_1^{\text{dif}}$:

$$g_1^{\text{comb}}(x) - g_1^{\text{dif}}(x) = \sum_{k=1}^{N} x^k \sum_{n\geq 0} \binom{n+sk-2}{n} \left((G_{\partial\rho}^{\text{comb}})^n - (G_{\partial\rho}^{\text{dif}})^n\right) \tilde{F}_k(\rho)$$

This is Theorem 3.4.13 for g^{comb} and is the Dyson-Schwinger equation itself for g_1^{dif}.

The first coefficient of g_1^{comb} and g_1^{dif} are the same, then applying the renormalization group equation resolved the bigger g_k to g_1 and for a fixed $g_{1,k}$ we can use the smaller $g_{1,k-l}$ to determine the equality recursively. Hence

$$g_1^{\text{comb}}(x) - g_1^{\text{dif}}(x) = 0$$

□

4 Beyond the chord diagram expansion

4.1 Using the chord-db and hunt for new numbers

There are a lot of new unknown number sequences hidden in rooted chord diagrams by ordering the terminal chords lexicographically and using the embedding that we discussed in 2.3.7 to build a direct limit. Here is a list of sequences:

1. Count the number of chord diagrams with given terminal set:

$$1, 3, 8, 15, 15, 30, 71, 105, 24, 51, 117, 180, 206, 315, 744, 945$$

2. The number of chord diagrams having branch left vectors of form $(0, 1, 1, \ldots, 1)$

$$1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 6, 3, 4, 1, 5, 4, 7, 3, 9, 5, 7, 2, 10, 6, 9, 3, 10, 4, 5, 1$$

which is closely related to the sequence OEIS: A002487:

$$1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, 1$$

Counting some chord diagrams and hunting for conjectures with chord-db:

```
sqlite> select count(*) from rccds where size=5 and branchv like "_ _ _ _ 4";
24
sqlite> select count(*) from rccds where size=5 and dvector like "%4";
24
sqlite> select count(*) from rccds where terminal="3 4 5" and dvector like "%3";
0
sqlite> select count(*) from rccds where terminal="3 4 5" and dvector like "%2";
2
sqlite> select count(*) from rccds where terminal="3 4 5" and branchv like "%2";
2
```

4.2 A canonical example

Let us consider $C := CW_n(\beta_1, \ldots, \beta_n)$ as defined already in [7]. For the sake of completness, we recall the definition:

Definition 4.2.1 (Cycloids, $CW_n(\beta_1, \ldots, \beta_n)$**.** A chord diagram where a chord i crosses only the chord $i + 1$ is called a cycloid. Let $\beta_1, \ldots, \beta_n$ non zero, then $CW_n(\beta_1, \ldots, \beta_n$ is

defined to be the wheel with n spokes and to the k-th chord of the wheel there is a cycloid of size β_k attached.

This is a big set of chord diagrams where we have good combinatorial control and all things we need can be calculated easily:

$$|C| = n + \sum_{k=1}^{n} \beta_k \tag{4.2.1}$$

$$\mathrm{ter}(C) = \{n + \beta_n, n + \beta_n + \beta_{n-1}, \ldots, n + \sum_{k=1}^{n} \beta_k\} \tag{4.2.2}$$

$$b(C) = n + \beta_n \tag{4.2.3}$$

$$a_{\delta(C,k)} = a_{n+\beta_n-k} a_{\beta_{n-1}} \cdots a_{\beta_1} a_0^{\sum_{k=1}^{n} \beta_k} \tag{4.2.4}$$

The root share decomposition is:

$$C = CW_1(\beta_1) \circ_{n-1} CW_{n-1}(\beta_2, \ldots, \beta_n) \tag{4.2.5}$$

And the only components of the corresponding branch left vector $\nu(C)$ that are non-zero are:

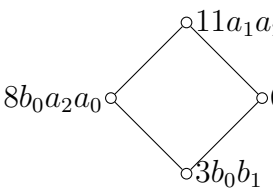
$$\nu_{n+\beta_n+\ldots+\beta_{n-N}}(C) = \begin{cases} \beta_{n-N} + 1 & N < n-1 \\ \beta_1 & N = n-1 \end{cases} \text{ where } 0 < N \leq n-1$$

Note that if we allow βs to be zero, then the latter equation is wrong, however the formulas 4.2.1 - 4.2.5 still hold true. However, if we allow βs to be zero, the representation is not unique: $CW_2(0,0) = CW_1(1)$.

4.3 Nice structures in the generic mixed case

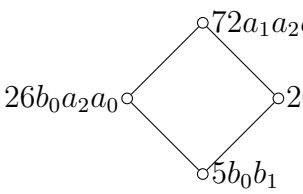
Lets consider the case, where $s = 2, N = 2$ which we call the generic mixed case. Then we have a lattice Sudoku game: Take a monomial from $\gamma_{1,n}$ that contains only unknowns $a_k := a_{1,k}$ from the first Mellin transform series. Let $b_k := a_{2,k}$ then applying the transformation $a_0 a_n \mapsto b_{n-1}$ continuously as long as it is possible yields monomials of $\gamma_{1,n}$ if we forget for a moment the integer coefficients. However, there is a connection between the coefficients. Take for example the case g_{14} and the monomial $a_1 a_2 a_0^2$, we could draw the corresponding monomials in a lattice:

Example 4.3.1. when $s = 2$:



note that: $11 - (8 + 6) + 3 = 0$

For $s = 3$ for the same monomial we have:



note that: $72 - 2(26 + 20) + 4 \cdot 5 = 0$

And indeed we have:

$$\sum_n (-1)^n (s-1)^n L_n = 0$$

where n sums over the n-th height level of the lattice. Note to prove this statement, we could not just modify the second mellin transform series

$$F_2(\rho) = \frac{1}{\rho} \sum_{k \geq 0} a_o a_{k+1} \rho^k$$

, because it depends on the height level of the lattice how many transforms are applied. Since this observation is hard to prove, we start with a bit coarser statement:

Let us look at the following summands $\gamma_{1,n}^{[m]}$ out of the anomalous dimension γ_1:

$$\gamma_{1,n}^{[m]} := \sum_{||C||=n, ||C||_2=m} \omega_C$$

then for arbitrary $s \in \mathbb{Z}_{\geq 2}$:

$$\sum_{m \geq 0} (-1)^m (s-1)^m \gamma_{1,n}^{[m]} = 0$$

where $||C||_2 = m$ denotes that the occurence of the decoration 2 appears exactly m

times in C. For a small $\gamma_{1,n}$ this relation is easy to proof, by just subtracting polynomials:

$$\begin{aligned}
\gamma_{1,4}^{[0]} &= 5(s-1)^3 + 16\binom{s}{2}(s-1) + 6\binom{s+1}{3} \\
\gamma_{1,4}^{[1]} &= 2(2s-1)\,(s+3(s-1)) \\
\gamma_{1,4}^{[2]} &= 2s-1
\end{aligned}$$

So the first polynomials that we can actually calculate are

$$\begin{aligned}
\gamma_{1,1}^{[0]} &= 1 \\
\gamma_{1,2}^{[0]} &= s-1 \\
\gamma_{1,3}^{[0]} &= 3s^2-5s+2 \\
\gamma_{1,4}^{[0]} &= 14s^3-31s^2+22s-5 \\
\gamma_{1,5}^{[0]} &= 84s^4-231s^3+224s^2-91s+14 \\
\gamma_{1,6}^{[0]} &= 620s^5-1946s^4+2340s^3-1364s^2+392s-42 \\
\gamma_{1,7}^{[0]} &= 5236s^6-18422s^5+26158s^4-19196s^3+7678s^2-1586s+132 \\
\gamma_{1,8}^{[0]} &= 49680s^7-192660s^6+310792s^5-270545s^4+137292s^3-40606s^2+6476s-429 \\
\gamma_{1,9}^{[0]} &= 21721s^8-2195721s^7+3931179s^6-3914509s^5+2372483s^4-896309s^3 \\
&\quad +206059s^2-26333s+1430 \\
\gamma_{1,10}^{[0]]} &= 5994155\,s^9-27052801\,s^8+52846049\,s^7-58712131\,s^6+40916811\,s^5 \\
&\quad -18557169\,s^4+5477263\,s^3-1014077\,s^2+106762\,s-4862
\end{aligned}$$

$$\begin{aligned}
\gamma_{1,4}^{[1]} &= 16\,s^2-20\,s+6 \\
\gamma_{1,5}^{[1]} &= 110\,s^3-194\,s^2+110\,s-20 \\
\gamma_{1,6}^{[1]} &= 894\,s^4-1990\,s^3+1608\,s^2-558\,s+70 \\
\gamma_{1,7}^{[1]} &= 8253\,s^5-21846\,s^4+22459\,s^3-11196\,s^2+2702\,s-252 \\
\gamma_{1,8}^{[1]} &= 84616\,s^6-256642\,s^5+315712\,s^4-201490\,s^3+70288\,s^2-12688\,s+924 \\
\gamma_{1,9}^{[1]} &= 949950\,s^7-3218522\,s^6+4558890\,s^5-3498530\,s^4+1569996\,s^3 \\
&\quad -411596\,s^2+58284\,s-3432 \\
\gamma_{1,10}^{[1]} &= 11565150\,s^8-42970140\,s^7+68258512\,s^6-60550398\,s^5+32799176\,s^4 \\
&\quad -11103316\,s^3+2291796\,s^2-263330\,s+12870
\end{aligned}$$

$$\gamma_{1,4}^{[2]} = 2s - 1$$
$$\gamma_{1,5}^{[2]} = 25\,s^2 - 25\,s + 6$$
$$\gamma_{1,6}^{[2]} = 285\,s^3 - 419\,s^2 + 198\,s - 30$$
$$\gamma_{1,7}^{[2]} = 3318\,s^4 - 6324\,s^3 + 4378\,s^2 - 1302\,s + 140$$
$$\gamma_{1,8}^{[2]} = 40464\,s^5 - 93444\,s^4 + 83912\,s^3 - 36570\,s^2 + 7720\,s + 630$$

$$\gamma_{1,6}^{[3]} = 12\,s^2 - 10\,s + 2$$
$$\gamma_{1,7}^{[3]} = 301\,s^3 - 380\,s^2 + 154\,s - 20$$
$$\gamma_{1,8}^{[3]} = 5640\,s^4 - 9426\,s^3 + 5716\,s^2 - 1488\,s + 140$$

and last but not least:

$$\gamma_{1,8}^{[4]} = 112\,s^3 - 124\,s^2 + 44\,s - 5$$

We guess immediately that the constant coefficients of $\gamma_{1,n}^{[0]]}$ are the Catalan numbers modulo sign. Also there are other number sequences related to the constant coefficients of $\gamma_{1,n}^{[1]}, \gamma_{1,n}^{[2]}, \gamma_{1,n}^{[3]}$ but because lack of data, we can not identify them uniquely.

And with the data given above, we can check that indeed:

$$\gamma_{1,8}^{[0]} - (s-1)\gamma_{1,8}^{[1]} + (s-1)^2\gamma_{1,8}^{[2]} - (s-1)^3\gamma_{1,8}^{[3]} + (s-1)^4\gamma_{1,8}^{[4]} = 0$$

Nevertheless, the non-constant coefficients of the polynomials $\gamma_{1,k}^{[l]}$ are hard to guess. Since it seems to be that $[s^0]\gamma_{1,k}^{[0]} = (-1)^{k+1}C_k$ where C_k is the k-th Catalan number, $[s^0]\gamma_{1,k}^{[1]} = (-1)^{k+1}\binom{2k}{k}$ and $[s^0]\gamma_{1,k}^{[2]} = (-1)^{k+1}\frac{(2k+1)!}{(k!)^2}$, we think that it might be a good idea to introduce Catalan like numbers in this business! They can be derived by the very classical moment problems.

4.4 Anomalous Dimension, moment problems and Catalan like numbers

The following section is the first try to connect the theory of analytic Dyson-Schwinger equations with the theory of moment problems. We only take the Hamburger moment into study, however, there are many more. For a first introductionary peek a good start might be the english Wikipedia pages . The Hamburger moment problem is stated as follows:

Definition 4.4.1. Given a sequence of real numbers $(a_n)_{n\in\mathbb{Z}_{\geq 0}}$ is said to solve the Hamburger moment problem if there exist a positive Borel measure μ:

$$a_n = \int_{-\infty}^{\infty} x^n d\mu(x)$$

The beauty of some moment problems is that there is a characterization by positive definitness of an operator that can be directly constructed by $(a_n)_{n\in\mathbb{Z}_{\geq 0}}$. In the case of the Hamburger moment problem we have the following theorem

Theorem 4.4.2. *(a_n) solve the Hamburger moment problem iff the following Hankel matrices $H^{(n)}$ are positive definite:*

$$H^{(n)}_{k,l} = (a_{k+l-2})_{k+l-2\leq n} \in \mathbb{R}^{(n+1)\times(n+1)}$$

i.e.

$H^{(0)} = (a_0)$, $H^{(1)} = \begin{pmatrix} a_0 & a_1 \\ a_1 & a_2 \end{pmatrix}$, $H^{(2)} = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 \end{pmatrix}$, ... *have there determinants positive.*

Furthermore, for (a_n) that solves the Hamburger Moment Problem there exists a unique positive Borel measure μ iff there are constants C, D such that for all

$$|a_n| \leq CD^n n!$$

Proof. See [9]. There the uniqueness proof use analytic vector techniques. There is a mistake in the proof, which is fixed in [10] (which was communicated to the author by Dr. Batu Güneysu). □

Example 4.4.3. The Catalan numbers are the unique number sequence $(C_n)_n$ that is defined by

$$\det(C_{k+l-2})_{k+l-2\leq n} = 1 \text{ and } \det(C_{k+l-1})_{k+l-1\leq n} = 1$$

The moment representation is:

$$C_n = \frac{1}{2\pi}\int_0^4 dx\, x^n \cdot \sqrt{\frac{4-x}{x}}$$

See wikipedia ([15]) for further references.

Inspired by the last example and similar relations on the determinants of the Hankel matrices of the Motzkin, Aigner defined the Catalan-like numbers.

In this section we want to develop a connection between the anomalous dimension γ_1 of the root chord diagram expansion to the Catalan-like numbers defined in [1]. What Catalan-like numbers of type (a, s) are will be given later, before we want to explain the setting a little bit more carefully and so give a summary of Aigners article.

Definition 4.4.4 (Admissible matrix)**.** An admissible matrix is an infinite lower triangular matrix, where the diagional contains only ones. and for all entries:

$$\sum a_{m,k}a_{n,k} = a_{m+n,0}$$

The numbers $a_{n,0}$ are called the Catalan-like numbers for the matrix A.

Lemma 4.4.5. *1. An admissible matrix $A = (a_{n,k}$ is uniquely determined by the sequence $(a_{n+1,n})$, on the other hand to every sequence $(b_n)_n$ of real numbers there exist an admissible matrix $(a_{n+1,n}) = b_n$.*

2. Let $a_{n+1,n} = b_n$ and define $s_0 = b_0, s_n = b_n - b_{n-1}$, then we have

$$a_{n,k} = a_{n-1,k-1} + s_k a_{n-1,k} + a_{n-1,k+1} \text{ for } n \geq 1$$
$$a_{0,0} = 1, a_{0,k} = 0 \text{ for } k > 0$$

If $a_{n,k}$ satisfies the latter recursion, then $a_{n,k}$ describes an admissible matrix with $a_{n+1,n} = s_0 + \ldots + s_n$

If $\sigma = (s_n)$ then the admissible matrix described before yield the catalan like numbers of type σ.

Example 4.4.6. The Catalan numbers equal the Catalan-like numbers of type $(1, 2, 2, \ldots)$

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 9 & 5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
14 & 28 & 20 & 7 & 1 & 0 & 0 & 0 & 0 & 0 \\
42 & 90 & 75 & 35 & 9 & 1 & 0 & 0 & 0 & 0 \\
132 & 297 & 275 & 154 & 54 & 11 & 1 & 0 & 0 & 0 \\
429 & 1001 & 1001 & 637 & 273 & 77 & 13 & 1 & 0 & 0 \\
1430 & 3432 & 3640 & 2548 & 1260 & 440 & 104 & 15 & 1 & 0 \\
4862 & 11934 & 13260 & 9996 & 5508 & 2244 & 663 & 135 & 17 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}$$

We hope that some of the column describe coefficients of the s-polynomials that we investigated in the DSE-Sudoku game.

Definition 4.4.7 (Catalan like numbers of type (a, s))**.** The Catalan-like numbers of type (a, s) that are the Catalan-like numbers of type $(a, s, s, s, \ldots)$. They are described by the generating function (see Proposition 3 of [1]):

$$C^{a,s}(x) = \frac{1 - (2a - s)x - \sqrt{1 - 2sx + (s^2 - 4)x^2}}{2(s - a)x + 2(a^2 - as + 1)x^2}$$

The question is which Mellin transform gives Catalan-like number of type (a, s) in the anomalous dimension. By comparing

$$C^{a,s}(x) - 1 = \gamma_1(x)$$

we find for the first coefficents of the Mellin transform

$$a_0 = a$$
$$a_1 = \frac{a^2 + 1}{a}$$
$$a_2 = 1/3\,\frac{sa - 1}{a^3}$$
$$a_3 = -1/45\,\frac{5\,sa^3 - 3\,a^2s^2 - 8\,a^2 + 11\,sa - 8}{a^5}$$
$$a_4 = \frac{1}{4725}\,\frac{175\,a^5s - 343\,a^4 - 168\,a^4s^2 + 45\,a^3s^3 + 821\,sa^3 - 408\,a^2s^2 - 776\,a^2 + 856\,sa - 493}{a^7}$$
$$a_5 = -\frac{1}{496125a^9}\Big(6125\,a^7s - 7203\,a^6s^2 - 13328\,a^6 + 42366\,a^5s + 3195\,a^5s^3 - 36696\,a^4s^2\Big)$$

By setting $a = 1$ we find:

$$1, 2, \frac{1}{3}(s - 1), \frac{-1}{45}\left(-3\,s^2 + 16\,s - 16\right)$$

where $3s^2 - 16s + 16 = (3s - 4)(s - 4)$ For $C^{(1,2)}$ we have the Catalan numbers, for $C^{(1,1)}$ the Motzkin numbers hence we have:

$$(1, 1) : 1, 2, 0, \frac{16}{45}$$
$$(1, 2) : 1, 2, \frac{1}{3}, -\frac{4}{45}$$

We do not know how to evolve this experiment into a proper statement. However, all the constant coefficients of the $\gamma_{1,k}^{[l]}$ seems to be a Catalan-like number for all cases that we checked so far (the unknown s of the polynomial $\gamma_{1,k}^{[l]}$ has nothing to do with the parameter s of the Catalan-like number $C^{a,s}$. Nevertheless, it seems for us impossible to

make a similar statement for the non-constant coefficients.

5 Computer related computations

5.1 Documentation of the chord programs

5.1.1 Using the chord-db

The chord-db is a database that contains all rooted connected chord diagrams from size three (the chord diagrams for size one and two are not included, since they are trivial to calculate) to size eight. The database consists of one table *rccds* that has the following columns and sqlite3 datatypes listed below :

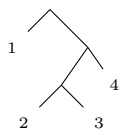
```
size: INTEGER
diagram: TEXT
terminal: TEXT
branchv: TEXT
dvector: TEXT
igraph: TEXT
rootshare: TEXT
instree: TEXT
```

It is useful for testing conjectures and recovering chord diagrams with several properties. For example you might want to know the root share decomposition of a given chord diagram without doing it by hand or you want to recover the chord diagram by a given insertion tree. We will discuss several examples here.

Example 5.1.1.

```
sqlite> select * from rccds where size=4 and instree = "(0 1 (0 (0 2 3) 4))";
```

queries the data for the chord diagram with the insertion tree:



The output of the sql query is will be the following cryptic output:

```
4|1 1 3 2 2 6 3 4 7 4 5 8|4|0 0 1 2|1 3 2 2|1(2) 2(3 4) 3(4) |(1 1 (2 2 (1 3 4)))|(0 1 (0 (0 2 3) 4))
```

which we will discuss now: Every column is seperated by |, so we have according to the column description that we gave already:

```
size = 4
diagram = 1 1 3 2 2 6 3 4 7 4 5 8
terminal = 4
branchv = 0 0 1 2
dvector = 1 3 2 2
igraph = 1(2) 2(3 4) 3(4)
rootshare = 1 1 (2 2 (1 3 4)))
instree = (0 1 (0 (0 2 3) 4))
```

The explanation of size, terminal and branchv is straight forward:

$$|C| = 4, \mathrm{Ter}(C) = \{4\}, \nu_C = (0, 0, 1, 2)$$

.

```
diagram = 1 1 3 2 2 6 3 4 7 4 5 8
```

encodes the chord diagrams in the following way: every third number encodes the label in the intersection order. After each label a pair of numbers tells us the chord. So the above chord diagram corresponds to the permutation:

$$(13)(26)(47)(58)$$

.

```
igraph = 1(2) 2(3 4) 3(4)
```

encodes the intersection graph by a simple adjacency list (since we have a bidirectional graph, we only need to encode the adjacency to higher vertices): The vertex 1 is adjacent to 2, 2 is adjacent to 3 and 4 and 3 is adjacent to 4. Now, we are able to explain

```
dvector = 1 3 2 2
```

It encodes the number of neighbors a vertex has. So in this example: 1 has one neighbor (the vertex 2), 2 has three neighbors (the vertices 1, 3 and 4), 3 has two neighbors (2 and 4) and 4 has also two neighbors (2 and 3).

```
rootshare = (1 1 (2 2 (1 3 4)))
```

tells us the root share decomposition into single chords which will have the induced labeling in Polish prefix notation. Thus,

$$(1\ 1\ (2\ 2\ (1\ 3\ 4)))$$

corresponds to

$$|_1 \circ_1 (|_2 \circ_2 (|_3 \circ_1 |_4))$$

where $|_k$ denotes the chord diagram with one chord and induced labeled with k.

Example 5.1.2. We can use the following sqlite notation to count several classes of chord diagrams. In sqlite a single underscore matches any single character, so

```
sqlite> select count(*) from rccds where branchv like "_ _ _ _ 4";
```

will count the chord diagrams

$$\{C \in \mathcal{R} : \nu_C = (\nu_1, \nu_2, \nu_3, 4)\}$$

which is 24. A % denotes in sqlite like statements arbitrary numbers of character, so the query

```
select count(*) from rccds where size=5 and dvector like "%4";
```

will ask for the chord diagrams of size 5 where in each chord diagram the last chord must cross four other chords. Again the answer is 24. This confirms the branch-left intersection graph duality (see 2.3.1).

5.1.2 Other programs

Since the chord program is written as parallel as it can get, the core is a function that generates (not necessary connected) rooted chord diagrams and passes it over to a callback function whenever it finds one. If we are interested in rooted connected chord diagrams we can directly test for it in the callback function:

```
if (is_connected (chord_diagram))
```

Note that you should be careful when defining your own callback function: It must be thread safe because it is possible that two callback functions are running parallel on different threads. So shared memory usage should be avoided. If you don't know the concepts of multi-thread programming just be sure that a callback functions never uses global variables except those passed into the arguments of the callback function. So if you want to calculate invariants of a chord diagram define a structure where all your invariants are stored in, allocate as many structures as the number of threads in your program and merge those structure at the end when all callback threads are joined.

A chord is encoded in the structure **chord_t**, where first is the label of the first vertex, second the label of the second vertex and id is reserved for a label in a chord diagram.

```
struct chord_t
{
  int first;
  int second;
  int id;
};
```

A special chord **EMPTY_CHORD** is defined and serves as delimiter object in a chord_t string(it has all three values to -1). Chords should be defined new by the method **mk_chord**. A chord diagram should be represented by a chord_t string (that is an chord_t array where at the end an EMPTY_CHORD determines the end in analogy to 0 in a plain c string. For example the following lines define the wheel with three spokes:

```
chord_t wheel[4];
wheel[0] = mk_chord(1,4,1);
wheel[1] = mk_chord(2,5,2);
wheel[2] = mk_chord(3,6,3);
wheel[3] = EMPTY_CHORD;
```

To generate a tikzpicture snippet for latex of a chord diagram you can use the program **plot-chord-diagram**. It is used as follows:

```
plot-chord-diagram "x0 y0 l0 x1 y1 l1 .... xN yN lN"
```

where x0 is the first vertex of the first chord, y0 the second vertex of the first chord and l0 its label and so on.

Example 5.1.3. `plot-chord-diagram "1 4 1 2 5 2 3 6 3"`
returns the following string that will display the wheel with three spokes in your latex file:

```
\begin{tikzpicture}
\draw[black,thin](0,0) circle(0.75);
\foreach \a in {0,60,...,359}
\draw[fill=black] (\a:0.75) circle(0.3mm);
\draw (0:0.75) circle(0.9mm);
\draw[thin,black](180:0.75) -- (0:0.75);
\draw(180:0.95) node{$1$};\draw[thin,black](240:0.75) -- (60:0.75);
\draw(240:0.95) node{$2$};\draw[thin,black](300:0.75) -- (120:0.75);
\draw(300:0.95) node{$3$};\end{tikzpicture}
```

For fast checking if a chord diagram is connected, you can use the function

```
bool fast_is_connected( chord_t* d, int size )
```

where d is a pointer to the beginning of a chord_t string and size gives the size of the chord diagram, not that you can save the EMPTY_CHORD delimiter since you have to specify the right size. The chord diagram size should not be bigger than the bit size of an int, so it should not be bigger than 8*sizeof(int). Example:

```
if( fast_is_connected(wheel,3) )
{
  printf("The wheel with three spokes is connected - what a surprise!");
}
```

To get the connected components of a disconnected chord diagrams you can use the function

```
void fast_get_connected_components( chord_t *dest,
                                    int *num,
                                    chord_t *src,
                                    int size )
```

where the value of the integer num will be number of connected components, src is the chord diagram that you want to be analyzed for connected components and size is its size. the pointer dest must point to a memory block where the connected components will be placed in, we recommend to allocate size*size*sizeof(chord_t) bytes for dest:

```
// defined wheel as wheel with three spokes as before
...
chord_t *dest = new chord_t[3*3];
int num = 0;
fast_get_connected_components( dest, &num, wheel, 3);
```

the connected components are saved in the following way as chord_t strings as before and are seperated via EMPTY_CHORD, since we have stored the number of connected components in num we can easily recover the connected components via a for loop.

To get the terminal chords and the right labeling you can use the method:

```
void fast_modify_to_intersection_order( chord_t *dest,
                                        int dest_size,
                                        chord_t* c,
                                        int size,
                                        int *label,
                                        int *terminals,
                                        int *num_term )
```

where **chord_t* c** is the source chord diagram, **size** the size of the source chord diagram **chord_t *dest** is the target chord diagram, textbfint dest_size the size of the target chord diagram, **int *label** pointer to an int that serves as a helper for the method, **int *terminals** a pointer to an int array of chord diagram size - 1 (because it is the maximal amount of terminal chords that a chord diagram can have) and **int *num_term** a pointer to an int where the number of terminals should be stored in.

Example 5.1.4.

```
  // defined wheel as wheel with three spokes as before
  // ...
int terminals[2];
int label;
int numter;
fast_modify_to_intersection_order( wheel, 3, wheel, 3, label, terminals, numter);
```

after execution the values of terminals[0] will be three, the value of terminals[1] will be undefined and the value of numter will be 1. The ids of the wheel chords will be set to the right labeling if they are not already set to it.

The following function calcultes the root share decomposition into two smaller chord diagrams $C = C_1 \circ_k C_2$:

```
void fast_rsd_decomp( chord_t* &dest1,
                      int* size1,
                      chord_t* &dest2,
                      int* size2,
                      int* index,
                      chord_t* src,
                      int src_size )
```

where C_1 will be stored in dest1 and $|C_1|$ will be stored in size1, C_2 will be stored in dest2 and $|C_2|$ will be stored in size2, k will be stored in the int where index points to, the input C needs to be placed into src and its size into src_size. Note that you need allocate the memory for dest1 and dest2 by yourself.

5.1.3 Documentation of bintree

bintree.cc and **bintree.h** implement a binary tree tree without dynamical memory allocation. It is useful for binary trees that are of bounded size as in our situation. Only allocating memory once, ensures that we do not waste cpu for allocating and deallocating memory for every node. The binary trees are restricted to a node value of an int but it should be no problem to modify the code to work with other types. Every node is modelled by the struct

```
struct bt {
  int data;
  bt* left;
  bt* right;
  bt* parent;
};
```

The following methods are implemented:

```
inline bt* bt_alloc(int size);
```

A macro for stdlib malloc to allocate *size* nodes and returning a pointer to the first node.

```
inline void bt_free(bt* ptr);
```

A macro for stdlib free.

```
inline bt* bt_new_node( bt** pool, bt* l, bt* r, bt* p, int d );
```

A macro for creating a new node where

pool points to an already allocated memory pool.

l points to the left node or NULL if it has no.

r points to the right node or NULL if it has no.

p points to the parent node or NULL if it has no.

d is the value of the node. If bt_new_node is used for a new memory pool the first node using it must be created it with $d = BT\_INIT\_VALUE$, The nodes *data* may be changed aftewards

Since we do not want dynamic memory allocation, the first argument must point to the memory pool that we allocated with bt_alloc or that is a fixed size bt array.

```
inline bool bt_is_leaf( bt* root );
```

A macro that checks if the left and right node of *root* is NULL which means that *root* is a leaf in our implementation.

```
inline void bt_copy( bt* dest, bt* src, int size );
```

A macro that wraps stdlib memcpy. Note that you have to ensure that dest is allocated properly by yourself.

```
void bt_pre_helper( bt* node, int nth, bt** target, int *pre_label );
```

After executing this method properly. *target* points to the *nth* pre-order node starting at *node*. *pre_ label* must point to an int that has been initialized with 1.

```
void bt_insert_at_nth_pre(bt** pool,bt** root, bt* src, int nth, int insert_data);
```

Insert the tree *src* at the pre-order *nth* node of *root* by creating a new node with *insert_ data*. For creating the new node the method needs a pointer *pool* which has at least one node allocated already.

```
void bt_print_pre( bt* root );
```

Prints a binary tree starting at *root* to stdio. Basically used for debug purposes.

```
void bt_print_node( bt* node );
```

Prints a tree *node* to stdio. Basically used for debug purposes.

5.1.4 Source code

The whole source code is available at github under the url:

```
http://github.com/Jeschli/rccds-cc
```

All chord diagram related algorithms are found in

```
rccds.cc resp. rccds.h
```

The algorithms are implemented in C/++. An example that generated all the data needed for the anomalous dimension is given in

```
lpthread-rccds-test.cc
```

Let's do a simple example to calculate all the data that we need of rooted connected chord diagrams of size $|C| = 9$.

```
g++ -DCHORDS=9 rccds.cc lpthread-rccds-test.cc -lpthread -lginac
./a.out
```

After about 95 seconds on a Schenker XMG C504 notebook with Fedora 22, the answer was a big table and the program finishes with the last line:

```
Total connected chord diagrams of size 9: 10401712
computation took about 95.00 second
```

We used all chord data up to size 10 to generate the polynomials of section 4.3. The source code is the following:

rccds.h

```
#ifndef RCCDS_H

#include <stdio.h>
#include <map>
#include <list>
#include <set>
#include <algorithm>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <vector>
#include <bitset>
#include "BinTree.h"

#include <string.h>

using namespace std;
using namespace markus;

/*
 * chord_t chord type
 */
  struct chord_t
  {
  int first;
  int second;
  int id;

  friend bool operator==(chord_t &c1, chord_t &c2);
  friend bool operator==(const chord_t &c1, const chord_t &c2);
  };

  chord_t mk_chord(int f,int s,int i);

  extern const chord_t EMPTY_CHORD; // = mk_chord(-1,-1,-1);

  typedef vector<chord_t> diagram_t;

  struct sort_vertex_pair
  {
  int *ptr;
  int n;
  };
```

```
  bool svpSmaller( sort_vertex_pair , sort_vertex_pair );

  bool crosses( chord_t, chord_t );
  bool fast_is_connected( chord_t*, int );
  void fast_remove_1( chord_t *, chord_t *, int );
  void fast_get_connected_components( chord_t *, int *, chord_t *, int );
  void fast_vertex_normalize( chord_t *, int );
  void fast_rsd_decomp( chord_t* &, int* , chord_t* &, int* , int* , chord_t* , int );
  /*
  Following method needs to be called like this:
  int label = 0;
  int num_term = 0;
  vector<int> terminals(CHORDS-1);

  get_intersection_terminals( chord_array, CHORDS, chord_array,
                              CHORDS, &label, &terminals, &num_term );
  */
  void get_intersection_terminals( chord_t *dest, int dest_size, chord_t* c,
       int size, int* label, vector<int> *terminals, int *num_term );

  typedef unsigned long long myint;
  void get_intersection_terminals_bitmask( chord_t   *dest, int dest_size,
       chord_t* c, int size, int *label, myint *bitmask );

  BinTree<int>* fast_build_int_tree( BinTreeManager<int>& , chord_t *, int );
  BinTree<int>* insert_at(BinTreeManager<int>&,  BinTree<int>* , BinTree<int>* , int );
  BinTree<int>* get_insertion_tree( BinTreeManager<int>&, BinTree<int>* );
  void get_ld_map( map<int,int> *, BinTree<int>*, int );

  void generate_diagrams( set<int> , diagram_t& , void *(*)(diagram_t*) );

  bool isSmaller( int i, int j);

  #ifndef CHORDS
    #define CHORDS 4
  #endif

#define RCCDS_H
#endif
```

rccds.cc

```
#include "rccds.h"
```

```
bool operator==(chord_t &c1, chord_t &c2)
{
  return (c1.first==c2.first) && (c1.second==c2.second) && (c1.id==c2.id);
}

bool operator==(const chord_t &c1, const chord_t &c2)
{
  return (c1.first==c2.first) && (c1.second==c2.second) && (c1.id==c2.id);
}

chord_t mk_chord(int f,int s,int i)
{
  chord_t rval;
  rval.first = f;
  rval.second = s;
  rval.id = i;
  return rval;
}

const chord_t EMPTY_CHORD = mk_chord(-1,-1,-1);

bool isSmaller( int i, int j) { return i<j; }

int min( set<int> s )
{
  return *min_element( s.begin(), s.end() , isSmaller );
}

bool svpSmaller( sort_vertex_pair s1, sort_vertex_pair s2 )
{
  return (s1.n<s2.n);
}


bool crosses( chord_t x1, chord_t x2 )
{
  return (
    (x1.first < x2.first) &&
    (x2.first < x1.second) &&
    (x1.second < x2.second)
  ) || (
    (x2.first < x1.first) &&
```

```
    (x1.first < x2.second) &&
    (x2.second < x1.second)
  );
}

/*
 * For fast processing the chord diagram should be a string
 * of size CHORD, it is assumed that it is this size! It also assumes
 * that bitsize(int) > CHORD > 1!
 */
bool fast_is_connected( chord_t* d, int size )
{
  int * disj_sets = new int [size]();
  disj_sets[0] = 1;

  for( int v = 1 ; v < size ; v++ )
    {
      int marked_sets = 0;

      for( int s = 0 ; disj_sets[s] != 0 && s < size ; s++ )
        {
          for( int w = 0 ; w <= v ; w++ )
            {
              int W = disj_sets[s] & ( 1 << w ); // test if the bit w
                                                 // is set
              if( !W ) continue;

              if( /*markus::*/crosses( d[v], d[w] ) )
                {
                  marked_sets |= ( 1 << s );
                  break;
                }
            }
        }
      if( marked_sets == 0 )
        {
          int s;
          for( s = 0 ; disj_sets[s] != 0 ; s++ ) {};
          disj_sets[s] = (1 << v);
        }
      else
        {
          int pos;
```

```
          int min;
          bool first = true;

          for( pos = 0 ; pos <= v ; pos++ )
            {
              if( marked_sets & (1<<pos) )
                {
                  if( first == true )
                    {
                      min = pos;
                      first = false;
                      continue;
                    }
                  disj_sets[min] |= disj_sets[pos];
                  disj_sets[pos] = 0;
                }
            }
          disj_sets[min] |= (1 << v);
        }
    }
  int s = 0;
  for( int k = 0 ; k < size ; k++ )
    {
      if( disj_sets[k] != 0 ) s++;
     }

  delete [] disj_sets;
  if( s == 1 ) return true;
  return false;
}


/*
  Removes chord 1 from diagram and normalizes the labels but not the ids!
  Requires a chord_t string of length N, dest must already be
  allocated with N-1. Writes the resulting chord string to dest
*/
void fast_remove_1( chord_t *dest, chord_t *src, int N )
{
  memcpy( dest, src+1, (N-1)*sizeof(chord_t) );
}

/* fast_get_connected_components(chord_t*,int*,chord_t*,int=)
```

```
 * In: src, size
 * src - array of chord_t of size
 * Out: num, dest
 * num - number of connected component ;
 *       this is not accurate if you set a bigger size
 *       than your chord diagram is actually is
 * dest - array of chord_t of size*size ; must be allocated before!
 */
void fast_get_connected_components( chord_t *dest, int *num, chord_t *src, int size )
{
  //  int disj_sets[size] = { 0 };
  int *disj_sets = new int[size]();
  disj_sets[0] = 1;
  for( int v = 1 ; v < size ; v++ )
    {
      int marked_sets = 0;
      for( int s = 0 ; disj_sets[s] != 0 && s < size ; s++ )
        {
          for( int w = 0 ; w <= v ; w++ )
            {
              int W = disj_sets[s] & ( 1 << w ); // test if the bit w
                                                 // is set
              if( !W ) continue;

              if( /*markus::*/crosses( src[v], src[w] ) )
                {
                  marked_sets |= ( 1 << s );
                  break;
                }
            }
        }
      if( marked_sets == 0 )
        {
          int s;
          for( s = 0 ; disj_sets[s] != 0 ; s++ ) {};
          disj_sets[s] = (1 << v);
        }
      else
        {
          int pos;
          int min;
          bool first = true;
          for( pos = 0 ; pos <= v ; pos++ )
```

```
            {
              if( marked_sets & (1<<pos) )
                {
                  if( first == true )
                    {
                      min = pos;
                      first = false;
                      continue;
                    }
                  disj_sets[min] |= disj_sets[pos];
                  disj_sets[pos] = 0;
                }
            }
          disj_sets[min] |= (1 << v);
        }
    }
  int s = 0;
  for( int k = 0 ; k < size ; k++ )
    {
      if( disj_sets[k] != 0 )
        {
          int t = 0;
          for( int m = 0 ; m < size ; m++ )
            {
              if( (disj_sets[k] & (1 << m)) )
                {
                  dest[ s*size + t] = src[m];
                  t++;
                }
            }
          if( t < size ) dest[ s*size + t ] = mk_chord(-1,-1,-1);
          s++;
        }
    }
  if( s < size ) dest[ s*size ] = mk_chord(-1,-1,-1);
  *num = s;

  delete [] disj_sets;
}

void fast_vertex_normalize( chord_t *src, int size )
{
  //  sort_vertex_pair * svp = new sort_vertex_pair[size*2];
```

```
  vector<sort_vertex_pair> svp(2*size);
  for( int i = 0 ; i < 2*size ; i++ )
    {
      if( i%2 )
        {
          svp[i].ptr = &src[i/2].second;
          svp[i].n = src[i/2].second;
        }
      else
        {
          svp[i].ptr = &src[i/2].first;
          svp[i].n = src[i/2].first;
        }
    }
  sort( svp.begin() , svp.end() , svpSmaller );
  for( int i = 0 ; i < 2*size ; i++ )
    {
      *svp[i].ptr = i+1;
    }
}

/*
 * fast_rsd_decomp
 */
void fast_rsd_decomp( chord_t* &dest1, int* size1,
                      chord_t* &dest2, int* size2,
                      int* index, chord_t* src, int src_size )
{
  if( src_size == 1 )
    {
      dest1 = src;
      *size1 = 1;
      dest2 = NULL;
      *size2 = 0;
      return;
    }
  chord_t* src_rem1 = new chord_t [src_size-1];
  chord_t* comp = new chord_t[(src_size-1)*(src_size-1)];
  memset(comp,0,(src_size-1)*(src_size-1)*sizeof(chord_t));
  int comp_num = 0;
  fast_remove_1( src_rem1, src, src_size );
  fast_get_connected_components(comp,&comp_num,src_rem1,src_size-1);
  if( comp_num == 1 )
```

```
  {
    dest1 = new chord_t;
    memcpy( dest1, src, sizeof(chord_t));
    *size1 = 1;
    dest2 = new chord_t [src_size-1];
    memcpy( dest2, src_rem1, sizeof(chord_t)*(src_size-1)) ;
    *size2 = src_size-1;
    *index = dest1[0].second - 2;
  }
else
  {
    int comp_num = 1;
    chord_t * temp_dest1 = new chord_t[src_size-1]();
    chord_t * temp_dest2 = new chord_t[src_size-1]();
    int size_1 = 0;
    int size_2 = 0;
    temp_dest1[size_1++] = src[0];
    for( int i = 0 ; i < (src_size-1)*(src_size-1) ; i++ )
   {
     if( (i%(src_size-1))==0 && comp[i] == EMPTY_CHORD )
       break;
     if( (i%(src_size-1))==0 );
     if( comp[i] == EMPTY_CHORD ) {
     continue;
     }
     if( comp[i].id == 0 ) continue;
     if( (i/(src_size-1)) == 0 )
     {
       temp_dest2[size_2++] = comp[i];
     }
     else
     {
       temp_dest1[size_1++] = comp[i];
     }
   }
    dest1 = new chord_t[size_1];
    dest2 = new chord_t[size_2];

    memcpy( dest1, temp_dest1, sizeof(chord_t)*size_1 );
    memcpy( dest2, temp_dest2, sizeof(chord_t)*size_2 );

    *size1 = size_1;
    *size2 = size_2;
```

```
      int y1 = dest1[0].second;
      int x2 = dest1[1].first;
      int min = (y1<x2)?y1:x2;

      *index = min - 2;

      delete [] temp_dest1;
      delete [] temp_dest2;
    }
  delete [] src_rem1;
  delete [] comp;
}

// this is quite crappy and depends that fast_rem1
// and fast_get_connected_components
// do not call by reference but by value
void fast_modify_to_intersection_order(
  chord_t *dest, int dest_size, chord_t* c, int size,
  int *label, int *terminals, int *num_term
)
{
  if( c == NULL ) return;
  // find the right chord to modify the label
  for( int i = 0 ; i < dest_size ; i++ )
    {
      if( dest[i].first == c->first && dest[i].second == c->second )
{
  dest[i].id = *label;
  (*label)++;
  break;
}
    }
 // the last chord in intersection order of a connected component is
 // always a terminal chord
  if( size == 1 ) {
    terminals[(*num_term)++] = (*label)-1; // label already increased before
    return;
  }
  chord_t d[CHORDS] = {0};
  chord_t conn_matrix[(CHORDS-1)*(CHORDS-1)] = {0};
  chord_t* comps[CHORDS] = {0};
  int comp_sizes[CHORDS] = {0};
```

```
  fast_remove_1(d,c,size);
  int num_comp;
  fast_get_connected_components( conn_matrix, &num_comp, d, size-1 );
  // parsing through connected components

  if( num_comp == 1 )
    {
      fast_modify_to_intersection_order(dest,dest_size,d, size-1, label,terminals,num_term);
    }

  else
    {

      for( int i = 0 ; i < num_comp ; i++ )
{
  comps[i] = new chord_t [size];
}
      int actual_comp = 0;
      int actual_size = 0;

      for( int i = 0 ; i < (size-1)*(size-1) ; i++ )
{
  if( (i%(size))==0 && conn_matrix[i].id == -1 )
    {
      comp_sizes[actual_comp] = actual_size;
      break;
    }
  if( conn_matrix[i].id == -1 )
    {
      comp_sizes[actual_comp] = actual_size;
      actual_comp++;
      actual_size = 0;
      continue;
    }
  if( conn_matrix[i].id == 0 ) continue;
  comps[actual_comp][actual_size] = conn_matrix[i];
  actual_size++;
}


      for( int i = 0 ; i < num_comp ; i++ )
{
```

```
  if( comp_sizes[i] == 0 )
  {
                continue;
  }
  fast_modify_to_intersection_order( dest, dest_size,
               comps[i], comp_sizes[i], label,terminals,num_term );
}
    }
}


BinTree<int>* fast_build_int_tree( BinTreeManager<int>& btm, chord_t *src, int size )
{
  chord_t *first = NULL; chord_t *second = NULL;
  int size1 = 0; int size2 = 0;
  int index = 0;
  fast_vertex_normalize(src,size); // vertex normalize
  fast_rsd_decomp(first,&size1,second,&size2,&index,src,size);
  BinTree<int>* iroot = btm.mk_BinTree_ptr(NULL,NULL,NULL,index);
  BinTree<int>* lnode;
  BinTree<int>* rnode;
  if( size1 == 1 ) {
    lnode = btm.mk_BinTree_ptr(NULL,NULL,NULL,first[0].id);
  }
  else {
    lnode = fast_build_int_tree(btm,first,size1);
  }
  if( size2 == 1 ) {
    rnode = btm.mk_BinTree_ptr(NULL,NULL,NULL,second[0].id);
  }
  else {
    rnode = fast_build_int_tree(btm,second,size2);
  }

  lnode->parent = iroot; rnode->parent = iroot;
  iroot->left = lnode; iroot->right = rnode;

// cleanup the memory allocated by fast_rsd_decomp
  if( first != NULL )
  {
    if( size1 == 1 )
      delete first;
    else {
```

```
      delete [] first;
      first = NULL;
    }
  }
  if( second != NULL )
  {
      delete [] second;
    second = NULL;
  }
  return iroot;
}


BinTree<int>* insert_at( BinTreeManager<int>& btm, BinTree<int>* left,
                         BinTree<int>* right, int n )
{
  BinTree<int>* rval = btm.copy(right);
  insert_at_nth_pre(btm,&rval,left,n,0);

  return rval;
}


BinTree<int>* get_insertion_tree( BinTreeManager<int>& btm, BinTree<int>* node )
{
  if( node == NULL )
    return NULL;
  if( isLeaf(node) ) {
   return btm.copy(node);
  }
  return insert_at(btm,
       get_insertion_tree(btm,node->left),
       get_insertion_tree(btm,node->right),
       node->data);
}


void get_intersection_terminals( chord_t *dest, int dest_size,
                                 chord_t* c, int size, int* label,
                                 vector<int> *terminals, int *num_term )
{
    if( size == 1 )
    {
```

```
        *label += 1;
        // fast_connected_component
        //  returns copies no references,
        //  we have to find the right reference manually :(
        for( int i = 0 ; i < dest_size ; i++ )
        {
            if( dest[i].first==c->first && dest[i].second==c->second)
            {
                dest[i].id = *label;
                (*terminals)[*num_term] = *label;
                *num_term += 1;
                break;
            }
        }

        return;
    }
    else
    {
        chord_t d[CHORDS] = {0};
        chord_t conn_matrix[(CHORDS-1)*(CHORDS-1)] = {0};
        chord_t* comps[CHORDS] = {0};
        int comp_sizes[CHORDS] = {0};
        int num_comp;

        *label += 1;

        for( int i = 0 ; i < dest_size ; i++ )
        {
            if( dest[i].first==c[0].first && dest[i].second==c[0].second )
            {
                dest[i].id = *label;
                break;
            }
        }
        fast_remove_1(d,c,size);
        fast_get_connected_components( conn_matrix, &num_comp, d, size-1 );
        for( int i = 0 ; i < num_comp ; i++ )
        {
            int mul = i*(size-1);
            if( conn_matrix[mul].id == -1 )
                break;
            int actual_size = 0;
```

```
            for( int j = 0 ; j < size-1 ; j++ )
            {
                if( conn_matrix[mul+j].id == -1 )
                    break;
                actual_size++;
            }
            get_intersection_terminals(dest,dest_size,
                                       &conn_matrix[mul], actual_size,
                                       label,terminals,num_term);
        }

    }
  }

void get_ld_map( map<int,int> *mp, BinTree<int>* node, int counter )
{
  if( node == NULL ) return;
  if( isLeaf<int>(node) )
    {
      pair<int,int> p;
      p.first = node->data;
      p.second = counter;
      mp->insert(p);
      return;
    }
  if( node->right != NULL )
    {
      get_ld_map(mp,node->right,++counter);
    }
  if( node->left != NULL )
    {
      counter = 0;
      get_ld_map(mp,node->left,counter);
    }

}
```

The following programming (lpthread-rccds-test.cc) generates the chord data using the lpthread library for multithreading.

```
#include "rccds.h"
#include <pthread.h>
```

```
#include <iostream>
#include "tbl_counter.h"

typedef vector<chord_t> diagram_t;

unsigned int total = 0;

struct thread_data_t
{
  set<int> A;
  diagram_t D;
  unsigned int total;
  TBLCounter tbl_counter;
} thread_data[(CHORDS*2-1)];

void *proc_diags( diagram_t* per,void* ptr )
{
  thread_data_t* data_ptr = (thread_data_t*) ptr;
  chord_t chord_array[CHORDS];
  int k = 1;

  for( diagram_t::iterator j = per->begin() ; j != per->end() ; j++ )
    {
      chord_t chord = *j;
      chord.id = k;
      chord_array[k-1] = chord;
      k++;
    }

  if( fast_is_connected(chord_array,CHORDS) )
  {
int label = 0;
   int num_term = 0;
   vector<int> terminals(CHORDS-1);
   get_intersection_terminals(chord_array,CHORDS,chord_array,CHORDS,
                      &label,&terminals,&num_term);
   data_ptr->total++;
               BinTreeManager<int> btm;
            BinTree<int>* int_tree = fast_build_int_tree( btm, chord_array, CHORDS );
          BinTree<int>* ins_tree = get_insertion_tree( btm , int_tree);
               map<int,int> bvector;
               get_ld_map( &bvector, ins_tree, 0 );
                data_ptr->tbl_counter.add( terminals, bvector );
```

```
  }

  return NULL;
}


void generate_diagrams_mthread( set<int> avail, diagram_t& temp_diagram,
     void *(*callback_fun)(diagram_t*,void*), void* ptr )
{
  if( avail.empty() )      // new diagram found, add it
    {
    (*callback_fun)(&temp_diagram,ptr);
      return;
    }

  set<int>::iterator min = min_element( avail.begin(), avail.end() , isSmaller );
  set<int>::iterator max = max_element( avail.begin(), avail.end() , isSmaller );
  for( int i = *(min)+2 ; i <= *max ; i++ )
    {
      if( avail.find(i) == avail.end() )
        {
          continue;
        }
      set<int> avail_new = avail;
      avail_new.erase(*min);
      avail_new.erase(i);
      diagram_t temp_diagram_new = temp_diagram;
      chord_t c;
      c.first= *min;
      c.second = i;
      temp_diagram_new.push_back( c );

      generate_diagrams_mthread( avail_new, temp_diagram_new, callback_fun, ptr );
    }
}



void *process_fun( void * ptr )
{
  thread_data_t* data = (thread_data_t*) ptr;
  generate_diagrams_mthread(data->A,data->D,&proc_diags,ptr);
  return NULL;
```

```
}


int main(int argc, char const *argv[])
{
  struct timespec tstart={0,0}, tend={0,0};
  clock_gettime(CLOCK_MONOTONIC, &tstart);

  set<int> avail;
  for( int i = 2 ; i <= 2*CHORDS ; i++ ) avail.insert(i);
  int k = 0;
  for( int i = 3 ; i <= (2*CHORDS - 1) ; i++ )
    {
      set<int> x = avail;
      x.erase(i);
      diagram_t d;
      chord_t first_chord = mk_chord(1,i,1);
      d.push_back(first_chord);
      thread_data[k].A = x;
      thread_data[k].D = d;
      thread_data[k].total= 0;
      k++;
    }

  pthread_t threads[(CHORDS*2-1)];
  for( int i = 0 ; i < (CHORDS*2-1) ; i++ )
    pthread_create( &threads[i],NULL,process_fun,(void*)&thread_data[i] );

  for( int i = 0 ; i < (CHORDS*2-1) ; i++ )
    pthread_join( threads[i], NULL );

  TBLCounter global;
  unsigned int total = 0;
  for( int i = 0 ; i < (CHORDS*2-1) ; i++ )
  {
    global.merge_from( thread_data[i].tbl_counter );
    total += thread_data[i].total;
  }
  global.out();
  cout << "Total connected chord diagrams of size " <<
       CHORDS <<  ": " << total << endl;
```

```
  clock_gettime(CLOCK_MONOTONIC, &tend);
  printf("computation took about %.2f seconds\n",
         ((double)tend.tv_sec + 1.0e-9*tend.tv_nsec) -
         ((double)tstart.tv_sec + 1.0e-9*tstart.tv_nsec));

return 0;
}
```

Bibliography

[1] Martin Aigner. Catalan-like numbers and determinants. *Journal of Combinatorial Theory, Series A*, 87(1):33 – 51, 1999.

[2] David J. Broadhurst and D. Kreimer. Combinatoric explosion of renormalization tamed by Hopf algebra: Thirty loop Pade-Borel resummation. *Phys.Lett.*, B475:63–70, 2000.

[3] L. Foissy. Faà di bruno subalgebras of the Hopf algebra of planar trees from combinatorial Dyson-Schwinger equations. *Adv. Math*, 218:136–162, 2008.

[4] Gerald B. Folland. *Quantum Field Theory: A Tourist Guide for Mathematicians*. American Mathematical Society, illustrated edition edition, August 2008.

[5] Dirk Kreimer. Dyson-Schwinger equations. `http://www2.mathematik.hu-berlin.de/~kreimer/wp-content/uploads/SkriptDSE.pdf`, 2013.

[6] Dirk Kreimer. Renormalization and renormalization group. `http://www2.mathematik.hu-berlin.de/~kreimer/wp-content/uploads/SkriptRGE.pdf`, 2013.

[7] N. Marie and K. Yeats. A chord diagram expansion coming from some Dyson-Schwinger equations. *ArXiv e-prints*, October 2012.

[8] Michael E. Peskin and Dan V. Schroeder. *An Introduction To Quantum Field Theory (Frontiers in Physics)*. Westview Press, 1995.

[9] M. Reed and B. Simon. *Methods of Modern Mathematical Physics: Fourier analysis, self-adjointness*. Number Vol. 2 in Fourier Nanlysis, Self-adjointness. Academic Press, 1975.

[10] Barry Simon. The Classical Moment Problem as a Self-Adjoint Finite Difference Operator. *Advances in Mathematics*, 137(1):82 – 203, 1998.

[11] David Tong. Lectures on quantum field theory. `http://www.damtp.cam.ac.uk/user/tong/qft.html`, 2006.

[12] Guillaume van Baalen, Dirk Kreimer, David Uminsky, and Karen Yeats. The QED β-function from global solutions to Dyson–Schwinger equations. *Annals of Physics*, 324(1):205–219, 2009.

[13] Walter D. Van Suijlekom. The hopf algebra of feynman graphs in Quantum Electrodynamics. *Letters in Mathematical Physics*, 77(3):265–281, 2006.

[14] Wikipedia. Binary tree — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Binary%20tree&oldid=650967650`, 2015. [Online; accessed 12-March-2015].

[15] Wikipedia. Catalan number — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Catalan_number&oldid=689506640`, 2015. [Online; accessed 23-November-2015].

[16] Karen Yeats. *Rearranging Dyson-Schwinger equations.* Memoirs of the American Mathematical Society, 2010.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Markus Hihn Berlin, den